Uporaba blockchain tehnologije u svrhu provjerljivosti podataka

Imrović, Mirna

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Mirna Imrović

UPORABA *BLOCKCHAIN* TEHNOLOGIJE U SVRHU PROVJERLJIVOSTI PODATAKA

Diplomski rad

Voditelj rada:
dr. sc. Ognjen Orel

Zagreb, rujan, 2023.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Svojima

Sadržaj

Sadržaj **iv**

Uvod **1**

1 Osnovni matematički i računarski koncepti **3**
1.1 *Hash* funkcije 3
1.2 Digitalni potpis 8
1.3 Distribuirani konsenzus 12

2 Pojam *blockchain*-a i primjene **15**
2.1 Struktura podataka 15
2.2 Modeliranje aplikacija temeljenih na *blockchain*-u 19
2.3 *Blockchain* kao tehnologija 21
2.4 Moguće primjene *blockchain*-a 22

3 Provjerljivost osobnih dokumenata pomoću *blockchain*-a **25**
3.1 Kombinirano korištenje *blockchain*-a i relacijske baze podataka 25
3.2 Hyperledger Fabric 27
3.3 Implementacija sustava za izdavanje diploma 30

Bibliografija **51**

Uvod

Blockchain tehnologija, odnosno tehnologija ulančanih blokova, u posljednjih je nekoliko godina dosta popularan pojam. Ponajviše se spominje u kontekstu kriptovaluta, kao centralna tehnologija koja omogućuje rad sustava za njihovo praćenje i razmjenu.

Promotri li se pobliže koje mogućnosti nudi implementacija *blockchain*-a u računalnom sustavu, može se uočiti da je ova tehnologija i šire primjenjiva. Provjerljivost i nepromjenjivost pohranjenog sadržaja osnovna su svojstva koja proizlaze iz same definicije strukture podataka *blockchain*. Povrh toga, sustavi temeljeni na *blockchain* tehnologiji mogu imati i brojne druge vrline, kao što su izostanak centralnog autoriteta nad podacima, automatizacija poslovne logike (koristeći pametne ugovore) i, što je najvažnije, svojstvo pohranjenih podataka da su javni, dok su njihovi vlasnici istovremeno anonimni. Svi ovi principi — bilo prošireni ili suženi — mogu biti vrlo korisni u raznim industrijama osim financija.

U ovom radu bavimo se mogućnošću korištenja *blockchain*-a kako bi se razvila digitalna usluga javne administracije. Posebno se osvrćemo na način kako se *blockchain*, kao nova tehnologija, može integrirati s već ustaljenom tehnologijom relacijskih baza podataka.

Konkretno, prvo izlažemo osnovnu matematičku i računarsku teoriju, pomoću koje potom definiramo *blockchain* i objašnjavamo modeliranje sustava temeljenih na *blockchain* tehnologiji. Na posljetku dajemo pregled praktičnog dijela rada, sustava koji na primjeru fakultetskih diploma ilustrira rad javno dostupne usluge kojoj pristupaju korisnici s raznim razinama vjerodajnica.

Poglavlje 1

Osnovni matematički i računarski koncepti

Prije nego počnemo govoriti o *blockchain*-u, dajemo pregled osnovnih tema računarstva i matematike koji su nužni za razvoj ideje o istome. Što se tiče matematičkih tema, promatramo kriptografske *hash* funkcije i digitalni potpis, a tema iz računarstva je konsenzus u distribuiranom sustavu.

1.1 *Hash* funkcije

U računarstvu ustaljene *hash* funkcije (hrv. funkcije raspršivanja) su surjekcije iz skupa podataka bilo kojeg tipa (u računarskom smislu) u skup $\{0, 1, \ldots, B-1\}$, pri čemu je B unaprijed fiksirani prirodni broj. One omogućuju raspoređivanje podataka iz domene u *pretince* radi lakše pretrage.

Zbog povećane potrebe za sigurnom komunikacijom preko nepouzdanih kanala, pojavljuju se i tzv. *kriptografske hash* funkcije. Naime, digitalna komunikacija nosi rizike od presretanja poruka ili izmjene njihovih sadržaja. Potrebno je osigurati integritet podataka i provjerljivost identiteta pošiljatelja. Kao alternativa standardnim kriptografskim praksama enkripcije i dekripcije poruka, budući da su algoritmi za iste često bili jednostavni za razbiti ili pretjerano neefikasni, pojavila se ideja da obje stranke u komunikacijskom kanalu svoje poruke sažmu i svedu na neprepoznatljiv oblik, i to tako da pokušaj otkrivanja originalne poruke bude vremenski neisplativ (bilo kome osim intendiranom primatelju). U takvoj je komunikaciji standardno primjenjivati i *digitalni potpis*, o kojemu ćemo više reći u sljedećem odjeljku, dok se zasad bavimo postupkom *sažimanja poruke*.

Definicija 1.1.1. *Neka je n fiksirani prirodni broj.* Kriptografska *hash* funkcija *je preslikavanje H koje za proizvoljni ulazni podatak (zapisan kao niz bitova) efikasno računa izlaz*

bitovne duljine n.

Dakle, ako je B skup svih konačnih nizova bitova, a B_n skup konačnih nizova bitova duljine n, onda je općenita kriptografska *hash* funkcija bilo koje preslikavanje iz B u B_n, čija je vremenska složenost dovoljno niska za praktičnu uporabu u modernim računalnim sustavima. Svojstvo ovih funkcija da se računaju efikasno u literaturi se naziva *jednostavnost izračunavanja* (engl. *ease of computation*). Drugo svojstvo, da su izlazi funkcije fiksne duljine, naziva se *kompresija* (engl. *compression*).

Valja napomenuti da nizove bitova na prirodan način poistovjećujemo s prirodnim brojevima u binarnom obliku ili bitovnim nizovima (stringovima). Stoga ćemo ulaze i izlaze *hash* funkcija najčešće zapisivati kao brojeve u heksadecimalnom obliku, imajući na umu da svaka znamenka predstavlja 4 bita binarnog zapisa. Koristimo termin *riječ* za bitovni string, niz bitova ili binarni broj, poistovjećujući ta tri pojma međusobno.

U ostatku rada koristit ćemo termin *hash* funkcija kao sinonim za kriptografsku *hash* funkciju. Izlaz *hash* funkcije nazivamo *hash* vrijednost ili sažetak ulazne poruke.

Neke *hash* funkcije zahtijevaju i tajni ključ kao dodatni ulazni podatak. Takve funkcije zvat ćemo *hash funkcije s tajnim ključem*[1].

Navodimo neka dodatna poželjna svojstva kriptografske *hash* funkcije $H : A \to B$, osim nužnih koja su navedena ranije:

- *Otpornost na prasliku*: za proizvoljni element $y \in B$, vrlo je teško pronaći element $x \in A$ takav da je $H(x) = y$.

- *Otpornost na drugu prasliku*: za proizvoljni $x \in A$, vrlo je teško naći neki drugi $x' \in A$ takav da je $H(x) = H(x')$.

- *Otpornost na podudaranja*: vrlo je teško pronaći dva različita elementa $x, x' \in A$ takva da je $H(x) = H(x')$.

U gornjem popisu svojstava koristi se sintagma *vrlo teško* kao skraćenica za:

> algoritam za navedeni postupak, ako postoji, dovoljno je visoke vremenske složenosti da se u praksi ne isplati pokušavati računati ga.

Hash funkcija koja je otporna na prasliku i na drugu prasliku naziva se *jednosmjerna hash funkcija*. *Hash* funkcija koja je otporna na drugu prasliku i na podudaranja zove se *hash funkcija otporna na podudaranja* (ili *jako jednosmjerna hash funkcija*).

[1] engl. *keyed hash functions*. Za *hash* funkcije bez tajnog ključa koristi se naziv *unkeyed hash functions*.

SHA-256

U svrhu demistifikacije *hash* funkcija, navodimo protokol za računanje jedne od najpoznatijih *hash* funkcija dosad: SHA-256. Radi se o standardu za *hash* funkcije postavljenom od strane američkog instituta *National Institute of Standards and Technology* (NIST) iz 2001. godine. SHA je skraćenica za *Secure Hash Algorithm*, a broj 256 označava da su svi izlazi te *hash* funkcije duljine 256 bitova.

Trenutno se vjeruje da je SHA-256 sigurna, odnosno da nema efikasnog algoritma za računanje njenog inverza i otporna je na drugu prasliku i podudaranja. Međutim, prethodnik ovog standarda, SHA-1, je kompromitiran — pronađena su dva različita ulaza s jednakom vrijednošću SHA-1 funkcije. Taj pronalazak bio je na modernom računalu vrlo zahtjevan za izračunati, ali ipak predstavlja sigurnosni rizik. Zbog toga postoji i noviji standard, Keccak, koji se smatra još težim za razbiti od SHA-256.

Neka je M ulazna poruka (podatak bilo kojeg tipa) koja je, bez smanjenja općenitosti, zapisana kao niz bitova. Duljina ulaza smije biti najviše 2^{64}. Za početak želimo osigurati da je duljina ulaza višekratnik broja 512 pa po potrebi dodajemo bitove na kraj ulaza. Ako je duljina ulaza l, dodajemo bit `1` na kraj i potom k bitova `0`, pri čemu je k najmanji nenegativni broj takav da je

$$l + 1 + k = 448 \pmod{512}.$$

Konačno, na sami kraj se dodaje 64-bitni binarni zapis broja l.

Potom se ulazna riječ rastavlja u N 512-bitnih riječi $M^{(1)}, \ldots, M^{(N)}$, od kojih je svaka dodatno rastavljena na 16 32-bitnih riječi $M_0^{(i)}, \ldots, M_{15}^{(i)}$.

Hash vrijednost računa se u N iteracija (rezultat svake označavamo s $H^{(i)}$), a rastavljena je na 8 riječi duljine 32 (j-tu riječ i-te iteracije označavamo $H_j^{(i)}$). Inicijalne vrijednosti su, s lijeva nadesno od $H_0^{(0)}$ do $H_7^{(0)}$:

```
6a09e667  bb67ae85  3c6ef372  a54ff53a
510e527f  9b05688c  1f83d9ab  5be0cd19
```

Dodatno, za računanje t-te iteracije bit će potrebna konstanta K_t. Navodimo $K_0, \ldots, K_{63}$ s lijeva nadesno:

```
428a2f98  71374491  b5c0fbcf  e9b5dba5  3956c25b  59f111f1  923f82a4  ab1c5ed5
d807aa98  12835b01  243185be  550c7dc3  72be5d74  80deb1fe  9bdc06a7  c19bf174
e49b69c1  efbe4786  0fc19dc6  240ca1cc  2de92c6f  4a7484aa  5cb0a9dc  76f988da
983e5152  a831c66d  b00327c8  bf597fc7  c6e00bf3  d5a79147  06ca6351  14292967
27b70a85  2e1b2138  4d2c6dfc  53380d13  650a7354  766a0abb  81c2c92e  92722c85
a2bfe8a1  a81a664b  c24b8b70  c76c51a3  d192e819  d6990624  f40e3585  106aa070
19a4c116  1e376c08  2748774c  34b0bcb5  391c0cb3  4ed8aa4a  5b9cca4f  682e6ff3
748f82ee  78a5636f  84c87814  8cc70208  90befffa  a4506ceb  bef9a3f7  c67178f2
```

Za algoritam su potrebne i bitovne operacije $\wedge$, $\vee$, $\oplus$ i $\neg$ nad 32-bitnim riječima. Pritom je $\oplus$ oznaka za ekskluzivno ili, tj.

$$x \oplus y \Leftrightarrow (x \wedge \neg y) \vee (\neg x \wedge y).$$

Koristimo i jezičnu operaciju konkateniranja riječi, oznaka $\|$. Zbrajanje brojeva simbolizirano je s + i podrazumijevamo da se obavlja modulo 2^{32}. Uvodimo oznaku SHR^n za bitovni pomak udesno za n, odnosno

$$SHR^n(x) = x \gg n.$$

Uvodimo i cirkularni pomak za n bitova,

$$ROTR^n(x) = (x \gg n) \vee (x \ll (32 - n)).$$

Za 32-bitne riječi x, y i z definiramo i sljedeće operacije:

$$\begin{aligned}
Ch(x, y, z) &= (x \wedge y) \oplus (\neg x \wedge z) \\
Maj(x, y, z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \\
\Sigma_0(x) &= ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \\
\Sigma_1(x) &= ROTR^2(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) \\
\sigma_0(x) &= ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x) \\
\sigma_1(x) &= ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)
\end{aligned}$$

Konačno, navodimo algoritam 1 koji računa funkciju SHA-256 na ulaznom podatku M nakon što su obavljeni početni koraci kao gore.

Algoritam 1: SHA-256

for $j = 1$ **to** N **do**
  for $i = 1$ **to** N **do**
    $a \leftarrow H_0^{(i-1)}$;
    $b \leftarrow H_1^{(i-1)}$;
    $c \leftarrow H_2^{(i-1)}$;
    $d \leftarrow H_3^{(i-1)}$;
    $e \leftarrow H_4^{(i-1)}$;
    $f \leftarrow H_5^{(i-1)}$;
    $g \leftarrow H_6^{(i-1)}$;
    $h \leftarrow H_7^{(i-1)}$;
    for $t = 0$ **to** 63 **do**
      if $t \leq 15$ **then**
        $W_t \leftarrow M_t^{(i)}$;
      else
        $W_t \leftarrow \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}$;
      $T_1 \leftarrow h + \Sigma_1(e) + Ch(e, f, g) + K_t + W_t$;
      $T_2 \leftarrow \Sigma_0(a) + Maj(a, b, c)$;
      $h \leftarrow g$;
      $g \leftarrow f$;
      $f \leftarrow e$;
      $e \leftarrow d + T_1$;
      $d \leftarrow c$;
      $c \leftarrow b$;
      $b \leftarrow a$;
      $a \leftarrow T_1 + T_2$;
    $H_0^{(i)} \leftarrow a + H_0^{(i-1)}$;
    $H_1^{(i)} \leftarrow b + H_1^{(i-1)}$;
    $H_2^{(i)} \leftarrow c + H_2^{(i-1)}$;
    $H_3^{(i)} \leftarrow d + H_3^{(i-1)}$;
    $H_4^{(i)} \leftarrow e + H_4^{(i-1)}$;
    $H_5^{(i)} \leftarrow f + H_5^{(i-1)}$;
    $H_6^{(i)} \leftarrow g + H_6^{(i-1)}$;
    $H_7^{(i)} \leftarrow h + H_7^{(i-1)}$;
return $H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)} \| H_7^{(N)}$;

1.2 Digitalni potpis

Digitalni potpis je mehanizam za potvrđivanje autentičnosti digitalnih podataka temeljen na modernoj kriptografiji. Konkretno, temelji se na kriptografiji javnog ključa, a svrha digitalnog potpisa je osigurati rješavanje najvažnijih problema moderne kriptografije u kontekstu komercijalne uporabe:

- *Povjerljivost*: poruku koju Alice pošalje Bobu ne može pročitati nitko treći;
- *Vjerodostojnost*: Bob zna da je samo Alice mogla poslati poruku;
- *Integritet*: Bob zna da Aliceina poruka nije bila izmijenjena prilikom slanja,
- *Nepobitnost*: Alice ne može naknadno zanijekati da je poslala poruku.

Tipični algoritam za digitalni potpis sastoji se od tri glavna koraka:

1. generiranje ključeva;
2. generiranje potpisa;
3. provjera potpisa.

Najpoznatiji algoritmi za digitalni potpis su DSA (*Digital Signature Algorithm*) i ECDSA (*Elliptic curve* DSA). Sustavi bazirani na *blockchain*-u mahom koriste ECDSA.

Definicija 1.2.1 (Eliptička krivulja)**.** *Neka je $\mathbb{K}$ polje i $\overline{\mathbb{K}}$ njegovo algebarsko zatvorenje.* Eliptička krivulja *E nad poljem $\mathbb{K}$ je skup rješenja jednadžbe*

$$F(x, y) = ax^3 + bx^2y + cxy^2 + dy^3 + ex^2 + fxy + gy^2 + hx + iy + j = 0$$

u projektivnoj ravnini $\mathbb{P}^2\left(\overline{\mathbb{K}}\right)$, pri čemu je $a, b, \ldots, j \in \mathbb{K}$. Pritom E mora biti nesingularna krivulja.

Eliptičku krivulju E nad poljem $\mathbb{K}$ označavamo s $E(\mathbb{K})$.

Kažemo da je krivulja nesingularna ako je u svakoj točki krivulje barem jedna parcijalna derivacija netrivijalna (različita od 0). Jednadžba iz definicije može se transformirati u *afinu Weierstrassovu formu*

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,$$

pri čemu je opet $a_1, a_2, a_3, a_4, a_6 \in \mathbb{K}$. Ako dodatno vrijedi $\text{char}(\mathbb{K}) \in \{2, 3\}$, onda se jednadžba može dalje transformirati u *kratku Weierstrassovu formu*

$$y^2 = x^3 + ax + b.$$

Ako nakratko promotrimo kubne eliptičke krivulje, umjesto njihovih dvodimenzionalnih projekcija kao u definiciji, možemo uvesti pojam racionalne točke. Točka P na krivulji $E(\mathbb{K})$ je $\mathbb{K}$-racionalna (ili, skraćeno, racionalna) ako postoji $\alpha \in \overline{\mathbb{K}}$ i točka $(x, y, z) \in \mathbb{K}^3 \setminus \{(0,0,0)\}$ tako da je

$$\alpha P = (x, y, z).$$

Može se pokazati da na eliptičkoj krivulji postoji klasa racionalnih točaka za koju vrijedi $z = 0$; ta je klasa reprezentirana točkom $(0, 1, 0)$, koju nazivamo *točkom u beskonačnosti*, a označavamo $\mathcal{O}$.

U daljnjem promatramo samo (dvodimenzionalne) eliptičke krivulje nad poljima karakteristike 2 ili 3. Stoga možemo eliptičku krivulju $E(\mathbb{K})$ shvaćati kao skup

$$\left\{(x, y) \in \mathbb{K} \times \mathbb{K} \;\middle|\; y^2 = x^3 + ax + b,\ a, b \in \mathbb{K},\ 4a^3 + 27b^2 \neq 0\right\} \cup \{\mathcal{O}\}.$$

Definiramo zbrajanje točaka (simbolizirano sa $+$) na eliptičkoj krivulji pomoću sljedećih formula:

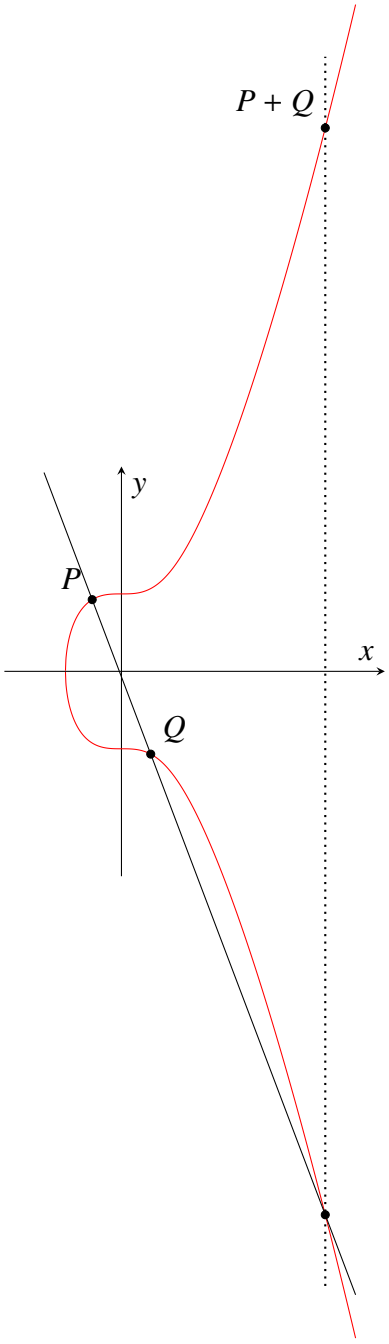
1. $-\mathcal{O} = \mathcal{O}$;
2. $-(x, y) = (x, -y)$;
3. $\mathcal{O} + (x, y) = (x, y)$;
4. ako je $Q = -P$, onda $P + Q = \mathcal{O}$;
5. ako je $P = (x_1, y_1)$, $Q = (x_2, y_2)$ i $Q \neq -P$, onda definiramo

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, & x_2 \neq x_1, \\ \dfrac{3x_1^2 + a}{2y_1}, & \text{inače}, \end{cases}$$
$$x_3 = \lambda^2 - x_1 - x_2,$$
$$y_3 = -y_1 + \lambda(x_1 - x_3)$$

i konačno, $P + Q = (x_3, y_3)$.

Može se pokazati da je $(E(\mathbb{K}), +)$ komutativna grupa. Dodatno, ako je $n \in \mathbb{N}$, onda definiramo

$$[n]P = \underbrace{P + \ldots + P}_{n \text{ puta}}.$$



Slika 1.1: Zbrajanje točaka na eliptičkoj krivulji

Primjer 1.2.2. *Slika 1.1 ilustrira zbrajanje točaka* $P = \left(-1, \sqrt{6}\right)$ *i* $Q = \left(1, -\sqrt{8}\right)$ *na*

eliptičkoj krivulji E, zadanoj nad poljem realnih brojeva, čija jednadžba glasi

$$y^2 = x^3 + 7.$$

Konačno, navedimo korake algoritma za digitalni potpis pomoću eliptičke krivulje prema ranije navedenim etapama. Uzimamo da Alice treba digitalno potpisati poruku $m \in \mathbb{N}$ koju šalje Bobu. Neka je $p \in \mathbb{N}$ i $\mathbb{F}_p$ konačno polje koje ima p elemenata. Dodatno, neka je E eliptička krivulja nad $\mathbb{F}_p$ i P neka točka prostog reda n na toj krivulji. Uzmimo da je H neka *hash* funkcija koju koriste i Alice i Bob.

Prvi je korak generiranje ključeva. Alice i Bob zasebno izvršavaju algoritam 2.

Algoritam 2: Generiranje ključeva

```
1 slučajno odaberi d ∈ {1, 2, ..., n − 1};
2 Q ← [d]P;
3 return javni ključ Q, tajni ključ d;
```

Potom Alice potpisuje svoju poruku, izvršavajući algoritam 3.

Algoritam 3: Generiranje potpisa

```
1  s ← 0;
2  do
3  |   r ← 0;
4  |   do
5  |   |   slučajno odaberi k ∈ {1, 2, ..., n − 1};
6  |   |   (x1, y1) ← [k]P;
7  |   |   r ← x1 (mod n);
8  |   while r = 0;
9  |   i ← k^{−1} (mod n);
10 |   s ← i · (H(m) + d · r) (mod n);
11 while s = 0;
12 return digitalni potpis (r, s) za poruku m;
```

Kako bi Bob provjerio da je poruku koju je primio zaista potpisala Alice, koristi algoritam 4. Uzima se da je Bobu poznat Alicein javni ključ Q, poruka m i pripadni potpis (r, s).

Vrijedi napomenuti da se u praksi najčešće koriste neke standardizirane *hash* funkcije, kao što su SHA-1, SHA-256 ili Keccak.

Algoritam 4: Provjera potpisa

provjeri: $r, s \in \{1, 2, \ldots, n-1\}$;
$w \leftarrow s^{-1} \pmod{n}$;
$h \leftarrow H(m)$;
$u_1 \leftarrow h \cdot w \pmod{n}$;
$u_2 \leftarrow r \cdot w \pmod{n}$;
$(x_0, y_0) \leftarrow [u_1]P + [u_2]Q$;
$v \leftarrow x_0 \pmod{n}$;
prihvati potpis akko vrijedi $v = r$;

1.3 Distribuirani konsenzus

Općenito, problem usuglašavanja u distribuiranom sustavu je problem dogovora njegovih procesa o nekoj informaciji. Najčešće se promatra pojednostavljena verzija ovog problema, tzv. problem *konsenzusa* u distribuiranom sustavu, koji se odnosi na dogovor procesa o vrijednosti jednog bita.

Ako pretpostavimo da sustav radi na sinkronoj mreži i da svi procesi rade korektno — bez ikakvih grešaka, onda je ovaj problem trivijalan. Greške koje se mogu javiti u pojedinom procesu su prestanak rada (*crash*), propust u radu (*omission*) i bizantinska greška. Bizantinske greške predstavljaju najtežu vrstu greške; pojedini proces može prestati komunicirati ili početi slati netočne ili maliciozne informacije.

Kažemo da je sustav koji radi na sinkronoj mreži *otporan na grešku* neke od navedenih vrsta ako se u njemu može postići konsenzus dok radi u uvjetima takvih grešaka. Kako bi sustav postao otporan na bizantinske greške, mora imati sljedeća svojstva:

- *Konačnost*: Svaki proces u konačnom vremenu donosi odluku o vrijednosti bita oko kojeg se mreža pokušava dogovoriti;
- *Valjanost*: Ako su svi korektni procesi predložili istu vrijednost bita, onda proizvoljni korektni proces mora donijeti odluku u skladu s tim prijedlogom;
- *Usuglašenost*: Svaki korektni proces odlučuje istu vrijednost bita.

Sustav koji je otporan na bizantinske greške također je otporan i na prestanak rada nekih procesa, kao i bilo kakve druge greške.

U slučaju da sustav radi na asinkronoj mreži i u prisustvu bilo kakvih grešaka, prema teoremu Fischera, Lynch i Patersona ne može se postići konsenzus.

Teorem 1.3.1 (Fischer–Lynch–Paterson)**.** *Ako je mreža asinkrona i dozvoljen je prestanak rada najviše jednog procesa, tada ne postoji algoritam za konsenzus koji istovremeno zadovoljava svojstva usuglašenosti, valjanosti i konačnosti.*

Kako bi se u modernim distribuiranim sustavima, koji mahom rade preko Interneta — inherentno asinkrone mreže, ipak uspjeli implementirati algoritmi za konsenzus, ugrađuju se dijelovi softvera koji služe za sinkronizaciju poruka koje se izmjenjuju u sustavu.

U sustavima koji koriste *blockchain*, predmet usuglašavanja je posljednje dodani blok u *blockchain*-u. Uobičajeno se koristi *implicitni konsenzus*, o kojemu ćemo više reći u nastavku rada, a čija se ideja temelji na tome da pojedini proces potvrđuje da se slaže s najnovijim blokom tako da nastavi graditi *blockchain* na njemu (odnosno, svoj predloženi blok veže na njega).

Poglavlje 2

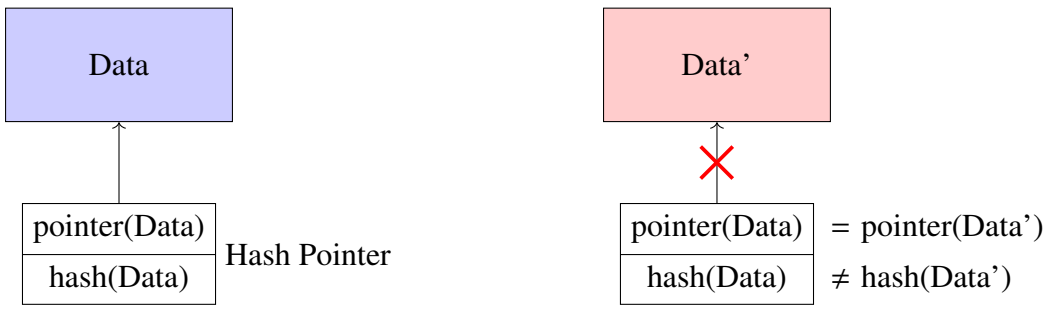
Pojam *blockchain*-a i primjene

U poglavlju 2 počinjemo se baviti *blockchain*-om. Prvo definiramo strukturu podataka — lanac blokova, a potom navodimo tipične načine na koje se modelira računalni sustav temeljen na *blockchain*-u. Objašnjavamo sintagmu *blockchain tehnologija*. Nabrajamo *tradicionalne* i neke novije moguće primjene *blockchain*-a u modernim sustavima.

2.1 Struktura podataka

U ovom odjeljku opisat ćemo najvažnije elemente *blockchain*-a kao strukture podataka. Pomoću njih ćemo zatim definirati i sami *blockchain* kao strukturu podataka.

Definicija 2.1.1. *Hash pointer za dani podatak x je pokazivač na x konkateniran sa $H(x)$, pri čemu je H neka kriptografska* hash *funkcija.*



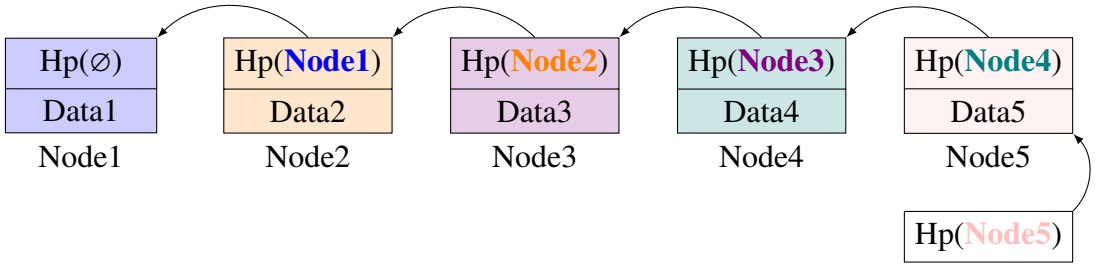
(a) *Hash pointer* na originalni podatak

(b) Ako je podatak izmijenjen, stari *hash pointer* više ne pokazuje na njega

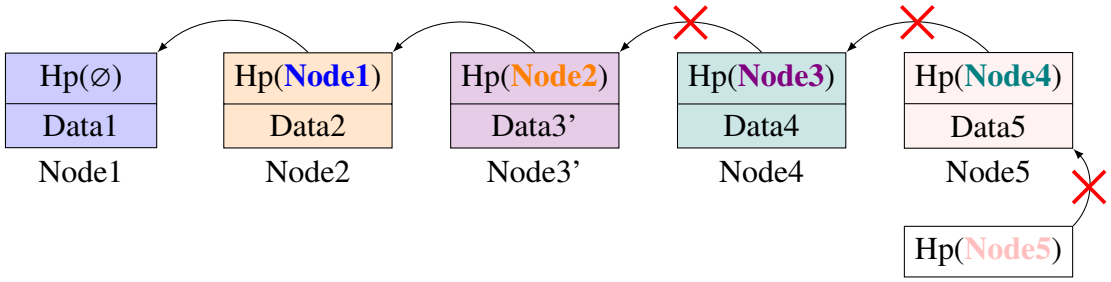
Slika 2.1: Vizualizacija *hash pointer*-a

Zbog ranije navedenih svojstava kriptografskih *hash* funkcija, bilo kakva perturbacija podatka x dovest će do toga da ranije izračunati *hash pointer* više ne pokazuje na taj podatak!

Nadalje, ako je niz podataka spremljen u vezanu listu u kojoj su rabljeni *hash pointer*-i, perturbacija u bilo kojem ranijem podatku će dovesti do greške koja će biti propagirana do kraja liste te ćemo time moći detektirati da je došlo do pokušaja da se podaci izmijene.



(a) Lista s originalnim podacima



(b) Lista s izmijenjenim podatkom

Slika 2.2: Promjena na podatku u vezanoj listi se može detektirati na kraju liste

Slika 2.2 prikazuje kako, ako neki korisnik posjeduje samo *hash pointer* na zadnji podatak u vezanoj listi, može detektirati promjenu podatka u bilo kojem ranijem podatku iz te liste.

Definicija 2.1.2 (Merkleovo[1] stablo)**.** *Neka je dan skup podataka S koje želimo smjestiti u jedan blok i neka kriptografska hash funkcija H. Za proizvoljni podatak x označimo njegov hash pointer (dobiven pomoću funkcije H) sa $Hp(x)$.*

Uzmimo da su podaci iz S sortirani po proizvoljnom smislenom uređaju. Dobiveni niz podataka označimo sa $x_1, \ldots, x_n$ i imenujmo prvom razinom Merkleovog stabla. Sada ponavljamo:

1. *za svaki $i \leq n$ računamo $Hp(x_i)$;*

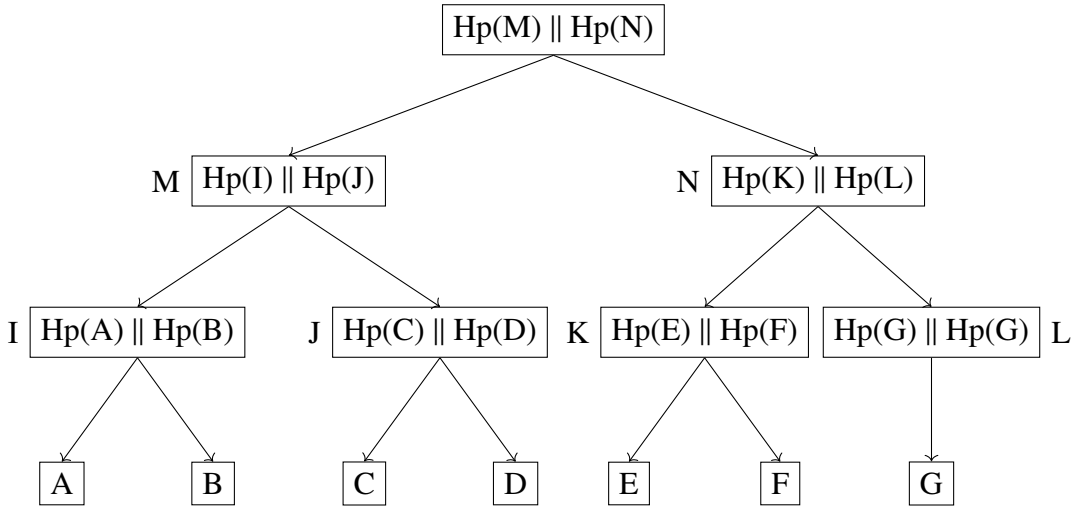
[1] Ralph Merkle, 1952.–, matematičar i računalni znanstvenik

2. *za svaki parni $j < n$ stvaramo novi podatak, koji je jednak $Hp(x_j)||Hp(x_{j+1})$. Ako je n neparan i veći od* 1*, onda će zadnji među novim podacima biti $Hp(x_n)||Hp(x_n)$;*

3. *ovako dobiveni niz podataka proglašavamo novom razinom Merkleovog stabla, a varijable n, $x_1, \ldots, x_n$ redefiniramo tako da ju opisuju;*

dok najnovija razina stabla ne bude duljine 1*. Taj element je* korijen Merkleovog stabla*. Prva razina čini* listove Merkleovog stabla*.*

Binarno stablo nastalo navedenim postupkom naziva se Merkleovo stablo*.*



Slika 2.3: Merkleovo stablo sa 7 listova

Visina Merkleovog stabla s n listova je $\lceil \log_2(n) \rceil$. Spomenuti uređaj podataka u listovima Merkleovog stabla u daljnjem označavamo s $\preceq$.

Sljedećim primjerom ilustriramo neka važna i korisna svojstva Merkleovog stabla.

Primjer 2.1.3 (Svojstva Merkleovog stabla)**.** *Pretpostavimo da Alice posjeduje pokazivač na korijen Merkleovog stabla i da joj Bob želi dokazati da je podatak x u jednom od listova tog stabla. Bob tada može Alice dokazati svoju tvrdnju tako da joj pokaže podatak x i sve blokove na putu od x do korijena — Alice će moći izračunati (u logaritamskom vremenu) sve* hash pointer*-e redom i krajnji rezultat usporediti s onim* hash pointer*-om koji posjeduje kako bi potvrdila Bobovu tvrdnju. Ovo svojstvo Merkleovog stabla naziva se* dokazivost prisutnosti *(*proof of membership*).*

S druge strane, uzmimo da Alice posjeduje cijelo Merkleovo stablo te da Bob lažno tvrdi kako je podatak x u nekom listu toga stabla. Tada Alice može opovrgnuti Bobovu tvrdnju na način da pronađe listove y, z između kojih bi se našao x kad bi stvarno bio

u stablu (tj. vrijedi $y \preceq x \preceq z$); potom može gore opisanom metodom dokazati njihovu prisutnost u stablu. Neistinitost Bobove tvrdnje slijedi iz činjenica da su y i z zaista listovi, da su neposredni susjedi u nizu listova, da je $y \preceq x \preceq z$. Opisano svojstvo Merkleovog stabla zove se dokazivost neprisutnosti *(proof of non-membership).*

Nadalje, pretpostavimo da Bob ima maliciozne namjere i pokušava izmijeniti neki podatak koji se nalazi u listu Aliceinog Merkleovog stabla. Tada će se pripadni hash pointer razlikovati od originalnoga pa će isto vrijediti za svaki sljedeći čvor na putu do korijena. Tako će i korijen postati različit od originalnoga pa će Alice moći detektirati pokušaj prijevare.

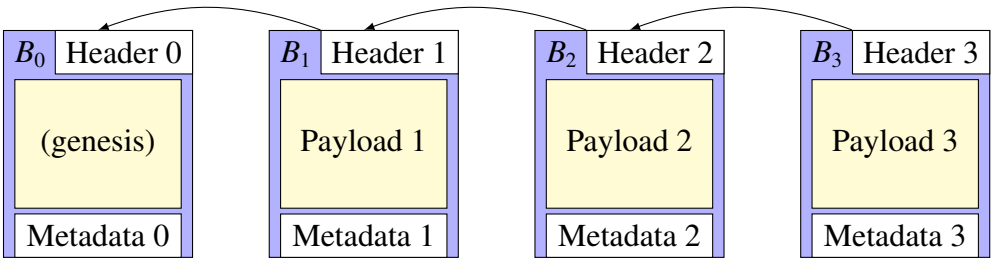
Merkleova stabla ne moraju imati sortirane listove. Međutim, ako izuzmemo tu pretpostavku iz definicije, onda se gubi vrlo korisno svojstvo dokazivosti neprisutnosti.

Konačno, pošto smo uveli sve potrebne tehničke pojmove, možemo precizno definirati strukturu podataka *blockchain* (hrv. lanac blokova). Općenito, *blockchain* je bilo koja lista vezana *hash pointer*-ima, kakvu smo opisali ranije u ovoj sekciji. Međutim, u svim praktičnim primjenama *blockchain*-a, pa tako i u ovom radu, blokovi u toj listi imaju ipak specifičniju strukturu.

Definicija 2.1.4. *Blockchain, iliti* lanac blokova*, je vezana lista nastala povezivanjem specijalnih blokova podataka pomoću hash pointer-a.*

Navedeni specijalni blokovi podataka imaju zaglavlje, korisni sadržaj (payload) i metapodatke. Pritom podaci koje želimo pohraniti na blockchain čine korisni sadržaj. Zaglavlje sadrži redni broj bloka, Merkleov korijen podataka iz trenutnog bloka i Merkleov korijen podataka iz prethodnog bloka.

Prvi blok u lancu, koji nema prethodnika, naziva se blok postanka*, odnosno genesis block.*



Slika 2.4: *Blockchain* s 4 bloka

2.2 Modeliranje aplikacija temeljenih na *blockchain*-u

Budući da se *blockchain* uglavnom koristi u sustavima za anonimizirane transakcije te nema centralnog autoriteta, sustavi temeljeni na *blockchain*-u dizajniraju se u modelu ravnopravnih partnera, odnosno kao *peer-to-peer* (P2P) sustavi.

Općenito, sustav građen u modelu ravnopravnih partnera ima veći broj istovrsnih procesa. Svaki proces naziva se partner (engl. *peer*). Svaki partnerski proces se izvršava na jednom računalu, a pojedini partneri su međusobno povezani komunikacijskim kanalima. Partner svoje dužnosti prema korisnicima obavlja uglavnom samostalno, ali u slučaju nedostatka informacija ili resursa može zatražiti iste od svojih susjeda u mreži. Pojedini partnerski proces istovremeno ima uloge i klijenta i poslužitelja u mreži — ovisno o potrebama. Ova vrsta mreže je visoko skalabilna čim je osigurana dobra replikacija podataka; procesi se mogu uključivati i isključivati iz mreže bez da cijeli sustav prestane s radom i bez da se izgube važni podaci koje je posjedovao samo ugašeni čvor.

Svaki čvor u P2P mreži, dakle, obavlja sličan posao i ima slične privilegije unutar sustava. Poslovi koje bi trebalo obavljati u sustavu temeljenom na *blockchain*-u su izvršavanje transakcija, osluškivanje i prosljeđivanje informacija o novim transakcijama, okupljanje određenog broja transakcija u novi blok te predlaganje istog za dodavanje u *blockchain* i naravno, utvrđivanje valjanosti zadnje dodanih blokova na *blockchain*. Ipak, ovisno o potrebama i mogućnostima korisnika koji održava pojedini čvor, taj se čvor može odreći ponekih odgovornosti, pa time i nagrada koje s njima dolaze. Drugim riječima, na planu čitavog sustava, mogu se uvesti razne kategorije čvorova, unutar svake od kojih čvorovi imaju različite razine odgovornosti. Primjerice, ako govorimo o sustavu za kriptovalute, čvorovima koji na sebe preuzmu više odgovornosti (konstantno skupljaju informacije o novim transakcijama i predlažu blokove) mogu se dati poticaji u obliku novih novčića te kriptovalute.

Dakle, funkcionalnost sustava koji koriste *blockchain* može se ugrubo rastaviti u 5 slojeva:

1. Sloj za konsenzus
2. Sloj za rudarenje
3. Sloj za propagiranje informacija
4. Semantički sloj
5. Aplikacijski sloj

Sloj za propagiranje informacija se bavi prosljeđivanjem *blockchain*-a (točnije, tzv. *ledger*-a — glavne knjige s podacima o transakcijama), kao i transakcija koje još nisu ugrađene u *blockchain*, kroz P2P mrežu. Najčešće se koristi jednostavni algoritam poput

algoritma poplavljivanja (engl. *flooding*); svaki čvor proslijedi informaciju svojim susjedima čim ju sazna.

Semantički sloj zadužen je za utvrđivanje ispravnosti prethodnih blokova, kao i obavljanje transakcija.

Konačno, aplikacijski sloj se bavi samom aplikacijskom logikom te izvršava pametne ugovore.

Neki sustavi, osim samog *blockchain*-a, održavaju i bazu podataka o stanjima sustava (engl. *state database*). Primjerice, jedan redak tablice te baze može opisivati stanje sustava nakon što se izvršila jedna transakcija.

Implicitni konsenzus

Sloj za konsenzus izvršava algoritam za distribuirani konsenzus, s ciljem utvrđivanja točnog stanja *blockchain*-a. Nerijetko se sustavi temeljeni na *blockchain*-u ustvari oslanjaju na *implicitni konsenzus*: ako je neki čvor dodao svoj novi predloženi blok na kraj *blockchain*-a, podrazumijevamo da nije našao nekonzistentnosti ili greške u prethodnim blokovima, odnosno da se složio da su oni ispravni. Najpopularniji algoritmi za konsenzus, pogotovo u sustavima za kriptovalute, su *Proof of Work* (PoW) i *Proof of Stake* (PoS).

U suštini, navedeni algoritmi za konsenzus služe tome da se odabere čvor koji će sastaviti i predložiti sljedeći blok na *blockchain*-u, odnosno koji će čvor biti vođa mreže u idućoj iteraciji ispitivanja konsenzusa. Oba algoritma opisujemo ukratko.

U PoW algoritmu promatra se količina uloženog rada u čvorovima koji se bave rudarenjem. Zadaje se kompleksni računski problem i onaj čvor koji prvi pronađe rješenje, osvojio je određenu (protokolom propisanu) količinu kriptovalute i pravo da predloži idući blok za *blockchain*.

PoS algoritam zahtijeva da čvorovi koji žele predložiti blok, predaju određenu (protokolom propisanu) količinu kriptovalute kao *kaparu*, koju dobivaju natrag ako blok bude prihvaćen od strane mreže, a inače ju gube.

Rudarenje

Rudarenje (*mining*) je postupak koji se uglavnom provodi u sustavima koji utiliziraju PoW algoritam za postizanje distribuiranog konsenzusa.

Smatra se da je u trenutku traženja novog bloka, cijeloj mreži poznata ciljana vrijednost t (*target*), kao i univerzalna *hash* funkcija H. Čvor koji predlaže blok sastavljen od transakcija $t_1, \ldots, t_m$ treba pronaći nekakvu proizvoljnu riječ n, tzv. *nonce*, tako da vrijedi

$$H(n \parallel \text{prethodni} \parallel t_1 \parallel \ \ldots \ \parallel t_m) < t,$$

pri čemu je

$$\text{prethodni} = \textit{hash} \text{ vrijednost prethodnog bloka},$$

a || označava jezičnu (bitovnu) konkatenaciju.

Navedeni računski zadatak naziva se *hash puzzle*. Zbog svojstva jednosmjernosti *hash* funkcije, nema poznatog efikasnog algoritma za računanje inverza od H, što čini ovaj zadatak prilično kompliciranim. Stoga se traženje *nonce*-a često svodi na pogađanje, a onaj čvor koji prvi uspije pogoditi je osvojio pravo da se njegov blok ugradi u *blockchain*.

Pametni ugovori

Pametni ugovori su dobar način da se automatizira dio aplikacijske logike u sustavu koji se temelji na *blockchain*-u. Radi se o računalnom programu koji implementira dogovoreni transakcijski protokol između određenih korisnika sustava.

U pravilu se sprema na *blockchain*, na način da se prevede u izvršivi kod i pošalje drugom korisniku u obliku transakcije. Naravno, kako je spremljen na *blockchain*, slijedi i da je nepromjenjiv. Glavna je značajka pametnih ugovora da se automatski izvršavaju kad god se obavi transakcija, pod uvjetima definiranim u ugovoru, između stranki koje su „potpisale" taj ugovor.

Budući da je pametni ugovor pohranjen na *blockchain*-u, tehnički bi trebao biti dostupan za uvid javno, odnosno svim korisnicima sustava. Neke mreže dopuštaju skrivanje ugovora od javnosti i definiranje vjerodajnica potrebnih da bi pojedini korisnik mogao vidjeti sadržaj ugovora.

U sustavu za kriptovalute, pametni ugovor može biti dobar način da se forsira transakcija novaca koje je jedan korisnik dužan drugome.

2.3 *Blockchain* kao tehnologija

Korištenje sintagme *blockchain tehnologija* je vrlo učestala pojava. U ovoj sekciji opisat ćemo zašto. Kako bismo lakše opisali taj termin, stavit ćemo *blockchain* u kontekst i opisati njegovu najrašireniju primjenu: sustave za kriptovalute.

Glavna ideja korištenja kriptovaluta je potpuno javna vidljivost transakcija između anonimnih korisnika. Kao i u gotovo svakom sustavu za razmjenu sredstava, transakcije se zapisuju u tzv. glavnu knjigu (engl. *ledger*). Nadalje, sustav za kriptovalute mora osigurati da napadač neće moći izmijeniti povijest transakcija. Drugi važni postulat sustava za kriptovalute je izostanak centralnog autoriteta (poput banke); sustavom i transakcijama upravljaju sami korisnici. Treće, nužno je da pojedini korisnik bude jedini koji ima moć da svoja sredstva pošalje na tuđi račun.

Iz svega navedenog, nameće se implementacija sustava za kriptovalute kao distribuiranog sustava (kao što smo već objasnili u prethodnom odjeljku). *Ledger* je zapisan na *blockchain* (svaka transakcija čini jedan podatak), u koji svaki korisnik može dobiti uvid — time je osigurano da su podaci o transakcijama javni i nepromjenjivi. Budući da nema

centralnog autoriteta, o valjanosti pojedine transakcije odlučuju svi korisnici zajedno, za što se koriste algoritmi za *distribuirani konsenzus*. Kako bi svaki korisnik mogao sigurno slati sredstva na druge račune, koristi se kriptografija javnog ključa, odnosno *digitalni potpis*.

Pokazuje se da sustavi koji podatke pohranjuju na *blockchain* i dodatno su obogaćeni navedenim konceptima iz računarstva i matematike, a pogotovo prošireni pametnim ugovorima, mogu imati brojne korisne primjene u modernizaciji poslovanja. Najpoznatije primjene su već navedeni sustavi za kriptovalute; može se vidjeti da je ova koncepcija i općenitije primjenjiva za sustave koji podržavaju bilo kakve transakcije, a ne samo transakcije sredstava plaćanja. Stoga je *blockchain* popularno prozvan tehnologijom, iako iz stručne perspektive takav naziv možemo smatrati ponajprije žargonskim.

2.4 Moguće primjene *blockchain*-a

Najpoznatija primjena *blockchain*-a, uz koju se koncept *blockchain*-a i proslavio, su već spomenute kriptovalute. Prva ozbiljnija kriptovaluta bio je Bitcoin, a kasnije se javljaju i alternativne valute poput Ethereuma i brojnih drugih.

Sustavi za kriptovalute bave se praćenjem transakcija te valute između svojih korisnika. Vidimo da je jedna instanca kriptovalute ustvari jedan vrlo jednostavan objekt, iliti token (u svijetu kriptovaluta najčešće se naziva novčić). Svi su tokeni međusobno jednaki; vlasniku jednog *bitcoin*-a nije važno je li to *bitcoin* A ili *bitcoin* B. Iz navedenog se pojavila ideja o uvođenju međusobno različitih tokena, od kojih bi svaki pojedini bio jedinstven. Radi se o konceptu *non-fungible token*-a (NFT). Korisnici su prepoznali NFT kao mogućnost za objavljivanje i utvrđivanje vlasništva digitalne umjetnosti pa su NFT-i postali nakratko jako popularni, nakon čega su opet podosta izgubili na vrijednosti.

U modernim sustavima za kriptovalute i NFT-e, transakcije se obavljaju na siguran i vrlo elegantan način, pogotovo ako su unaprijed uspostavljeni pametni ugovori. Takva promjena bila bi dobrodošla i u sustavima javne uprave, gdje se obavljaju transakcije dokumenata i informacija između tih ustanova i njihovih korisnika. Slično bi se moglo implementirati i u međusobnoj komunikaciji tvrtki, obrta i ostalih poduzeća.

Ako se razne vrste dokumenata modeliraju kao tokeni, a za korisnike se definiraju razne vrste vjerodajnica, onda se izdavanje dokumenata javne uprave može modelirati transakcijom koja se zapisuje na *blockchain*. Uočimo da dokumenti javne uprave nisu svi međusobno jednaki, ali nisu ni unikatni, nego pisani prema određenim obrascima; možemo reći da se radi o *parametriziranim tokenima*. Primjerice, ako se vlasnički list nekretnine modelira kao token u sustavu s *blockchain*-om, onda se promjena vlasnika pojedine nekretnine može vrlo lako pratiti kao transakcija tokena tipa „vlasnički list”, čiji pošiljatelj je trenutni, a primatelj sljedeći vlasnik te nekretnine.

S druge strane, moguće je tokenima modelirati i dokumente nad kojima nema smisla raditi transfer, ali koristiti *blockchain*-ovo svojstvo nepromjenjivosti kako bi podaci o vlasništvu takvih dokumenata bili javno dostupni. Jedan primjer takvog dokumenta je diploma o završenom fakultetu. Diploma, kada se kreira, zauvijek pripada istoj osobi, ali vlasništvo osobe nad diplomom je podatak koji treba biti transparentan i javno provjerljiv.

Vrijedi napomenuti da ideja korištenja *blockchain*-a za digitalizaciju birokratskog poslovanja nije potpuno nova i originalna, ali zasad nije ni ozbiljnije implementirana. Primjerice, europska inicijativa *European Blockchain Partnership* (EBP) pokrenula je projekt *European Blockchain Services Infrastructure* (EBSI), namijenjen za poboljšanje poslovnih procesa u javnoj upravi diljem Europe, a koji je još uvijek u razvoju. Čvorovima u toj mreži upravljaju posebna tijela koja imaju povjerenje države ili EBP-a. Isto tako, samo izdvojeni korisnici imaju pravo dodavati pametne ugovore. Važan dio funkcionalnosti EBSI-ja su provjerljive vjerodajnice, koje odabrane ustanove izdaju korisnicima. Korisnici mogu koristiti tzv. novčanik (engl. *wallet*), aplikaciju kroz koju imaju pregled svih svojih dokumenata i vjerodajnica.

U nastavku ovog rada bavimo se daljnjim istraživanjem ideje digitalizacije poslovanja javnih tijela pomoću *blockchain*-a.

Poglavlje 3

Provjerljivost osobnih dokumenata pomoću *blockchain*-a

U ovom poglavlju bavimo se predstavljanjem konkretne primjene *blockchain*-a za razmjenu javno dostupnih dokumenata između privatnih osoba i državnih institucija. Također i opisujemo implementaciju projekta nastalog u sklopu ovog diplomskog rada, koja ilustrira izdavanje i pregled fakultetskih diploma pomoću *blockchain*-a.

3.1 Kombinirano korištenje *blockchain*-a i relacijske baze podataka

Današnji informacijski sustavi uglavnom se oslanjaju na korištenje relacijskih baza podataka, što je vrlo efikasan način za pregled i manipulaciju velikog broja podataka organiziranih u tablice. Međutim, relacijske baze podataka same po sebi ne nude transparentan pregled promjena nad pojedinim podacima; jedini način da se to postigne je ručno pregledavanje zapisnika (engl. *log*-ova), koje u pravilu nije dostupno krajnjim korisnicima. Tako, primjerice, ako promatramo relaciju koja opisuje vlasnički list nekretnine, onda iz relacijske baze podataka nije jednostavno doći do podatka o promjeni vlasnika pojedine nekretnine, nego je potrebno pregledavati zapisnike ili unaprijed kreirati dodatnu relaciju koja opisuje promjenu vlasnika nekretnine.

Napomenimo da i neki sustavi izgrađeni nad *blockchain*-om interno koriste baze podataka kako bi pohranjivali stanja sustava između transakcija. Ovo, međutim, ne povlači nužno da poanta iz prethodnog paragrafa gubi na važnosti; naglasimo da se baze podataka koriste *interno*. To znači da, ukoliko neke organizacije pristanu svoju poslovnu komunikaciju obavljati koristeći *blockchain*, utoliko više neće biti nužno da svaka od njih pojedinačno implementira vlastitu podršku za pregled povijesti podataka. Pregled povijesti

time može biti efikasniji i jednostavniji za postići.

Za baze stanja sustava u pravilu se koriste NoSQL baze podataka, kao što su *key-value* baze ili dokumentske baze. Pojedinom tokenu se dodjeljuje jedinstveni identifikator (ključ), a sami token se pohranjuje u bazu kao vrijednost tog ključa — bilo u obliku niza bajtova ili JSON[1] objekta. Ipak, u ovom se poglavlju bavimo idejom paralelnog korištenja relacijske baze podataka i *blockchain* sustava, ne obazirući se na detalje implementacije *blockchain* sustava.

Druga, ali nipošto manje važna, prednost *blockchain* sustava bio bi *sustav povjerenja* koji je impliciran kad se govori o *tehnologiji blockchain*. Naime, sustavi javne uprave nominalno su transparentni, pod uvjetom da građani (i pravne osobe) imaju povjerenja u istinitost izvješća o poslovanju svake javne organizacije. Međutim, postoji mogućnost da se određene transakcije dogode bez znanja javnosti, dok u *blockchain* sustavima sumnjive transakcije dokumenata ne bi mogle proći bez odobrenja većine organizacija s povjerenjem (a ako bi i prošle, bile bi jasno vidljive na *blockchain*-u).

Još jedna vrlina sustava s *blockchain*-om, ako je šire primijenjen u javnoj upravi, jest mogućnost održavanja više odvojenih *ledger*-a koji prate više različitih vrsta dokumenata te njihova međusobna interakcija. Isto tako, korištenjem tzv. *wallet*-a, građanin također može na jednostavniji način interagirati s raznim sustavima kojima ima pristup.

Možemo, dakle, uočiti potencijalnu korisnost *blockchain*-a u svrhu praćenja javnih dokumenata. Za transfere dokumenata *blockchain* je elegantnije rješenje nego relacijska baza podataka, a također je prisutno i svojstvo nepromjenjivosti podataka pa je vlasništvo nad pojedinim dokumentom transparentno. Ipak, relacijske baze podataka su i dalje puno korisnije za čuvanje nekih osnovnih podataka koji trebaju ostati privatni ili za koje nije nužno da budu nepromjenjivi i transparentni. Stoga njihovo paralelno korištenje predstavlja optimalnu ideju za unapređenje digitalnog javnog poslovanja.

Projekt izrađen u sklopu ovog rada je jednostavni sustav za izradu i praćenje diploma fakulteta. Kreiran je objekt tipa `Diploma`, koji se koristi kao token u maloj *peer-to-peer* mreži koja održava *blockchain* s podacima o nastanku i izmjenama raznih diploma. Sama mreža ima dva čvora, od kojih svaki ima definirane vjerodajnice za po jednog administratora i jednog korisnika. Jedan čvor pripada organizaciji koja ima ovlasti da kreira i mijenja diplome na *blockchain*-u koristeći bazu podataka o studentima, a dizajnirana je i pripadna relacijska baza podataka s nekoliko osnovnih tablica koje su potrebne da bi se kreirala diploma. Drugi čvor pripada organizaciji koja nudi javni pregled postojećih diploma. Za svaki čvor implementirane su aplikacije koje se koriste kroz naredbeni redak. Primjerice, prva organizacija bi mogla biti fakultet, a korisnik koji rabi pripadnu aplikaciju ovlaštena osoba s pristupom relacijskoj bazi podataka (kao što je npr. baza podataka Informacijskog sustava visokih učilišta — ISVU); druga organizacija bi mogla biti javno dostupna usluga

[1] *JavaScript Object Notation* (JSON) je jednostavni, ljudima čitljiv format tekstualne datoteke koji se najčešće koristi u komunikaciji *web* servera i aplikacija.

koja omogućuje pregled javnih podataka uz autentikaciju preko usluge e-Građani. Projekt je izrađen koristeći programski okvir (engl. *framework*) Hyperledger Fabric. U narednim odjeljcima opisujemo Fabric i samu implementaciju projekta.

3.2 Hyperledger Fabric

Hyperledger je projekt zaklade Linux čija misija je razvoj raznih sustava nad *blockchain*-om sa svrhom vođenja poslovanja u raznim industrijama. Radi se na definiranju više različitih *blockchain* standarda, svaki od kojih ima svoju primjenu, a pritom su otvoreni za komunikaciju i suradnju s javnosti — svi projekti su otvorenog koda. Jedan od Hyperledgerovih projekata je Hyperledger Fabric, koji razvija sustave nad *blockchain*-om koji su privatni i u kojima mogu sudjelovati samo određeni akteri s povjerenjem ostalih aktera u sustavu ili nekog vanjskog autoriteta (engl. *permissioned system*).

Djelomična centralizacija autoriteta

Zbog ograničenosti pristupa sustavu, Fabric malo odudara od ustaljene ideje koja se primjenjuje kod sustava s *blockchain*-om. Naime, u ranijem opisu tipičnih sustava temeljenih na *blockchain*-u, istaknuli smo, između ostalog, potrebu za izostankom centraliziranog autoriteta. Međutim, uvođenje djelomične centralizacije ima i određene benefite. Kao prvo, može se postići ograničen pristup javnim podacima koristeći razne vrste vjerodajnica (npr. može se postići da neke podatke smiju vidjeti samo građani Republike Hrvatske, ali ne i bilo tko u svijetu). Kao drugo, gubi se potreba za traženjem konsenzusa o stanju na *blockchain*-u; članovi sustava mogu dati povjerenje jednom od čvorova da uvijek vodi računa o stanju na *blockchain*-u i ne obavlja nikakve druge zadatke. Vrijedi napomenuti da je konsenzus o valjanosti pojedine transakcije i dalje nužan — kažemo da čvorovi trebaju dati svoje odobrenje (engl. *endorsement*) za pojedinu transakciju.

Kako bi se postigla navedena privatnost mreže, koriste se posebni servisi za izdavanje vjerodajnica i za čitanje pojedine vjerodajnice da bi se njenom vlasniku odobrio pristup mreži. Prvi servis naziva se ustanova za izdavanje vjerodajnica — *Certificate Authority* (CA), a drugi *Membership Service Provider* (MSP). Umjesto Hyperledgerovog CA, može se koristiti i neka druga organizacija, poput DigiCerta ili GoDaddyja.

Ostale posebnosti Fabrica

Unatoč navedenoj velikoj razlici u usporedbi s uobičajenim *blockchain* sustavima, Fabric ima sve ostale tipične elemente *blockchain* sustava. Definira se jedno ili više sredstava (tokena) nad kojima se vrše transakcije, same transakcije bilježe se u glavnoj knjizi koja je zapisana na *blockchain*, implementiraju se pametni ugovori (koji se u Fabricu često

nazivaju i *chaincode*) radi automatizacije transakcijske logike, izvršavaju se transakcije o čijoj se valjanosti treba postići konsenzus itd.

U Fabricu je definiranje glavnog *chaincode*-a na svakom kanalu nužno da bi se uopće mogle obavljati bilo kakve transakcije.

Istaknimo još nekoliko inovacija koje Fabric uvodi. Omogućeno je stvaranje zasebnih skupina čvorova iz mreže, tzv. podmreža, koje između sebe tvore *kanal* na kojemu održavaju vlastiti *ledger*. Podmrežu nerijetko čine oni čvorovi koji pripadaju nekoj organizaciji pa organizacija ima svoje interne kanale, kao i kanale pomoću kojih komunicira s drugim organizacijama. *Ledger* nekog kanala može imati zapise o transakcijama jedne ili više vrsta sredstava. Štoviše, da bi se *ledger* uopće uspostavio, prvo mora biti definiran kanal između dva ili više čvorova.

Druga inovacija kojom se Fabric ističe je mogućnost održavanja *kolekcija privatnih podataka* na pojedinom kanalu. Transakcije privatnih podataka isto su zapisane na *ledger*, ali svi čvorovi osim pošiljatelja i intendiranih primatelja mogu vidjeti samo *hash* vrijednosti tih podataka.

Još jedna posebnost Fabrica je mogućnost definiranja politika koje propisuju čije odobrenje je potrebno da bi se definirala konfiguracija kanala ili pojedina transakcijska logika.

Glavna knjiga i stvaranje *blockchain*-a

Fabric specificira da se glavna knjiga (engl. *ledger*) sastoji od *stanja sredstava* i *blockchain*-a. Stanje sredstava je baza podataka koja sadrži podatke o trenutnim vrijednostima atributa za sva sredstva koja su prisutna u bazi. Omogućeno je korištenje LevelDB (*key-value*) ili CouchDB (dokumentske) baze za ovu svrhu. Za stanje sredstava se u Fabricovoj dokumentaciji koristi termin *world state*. Drugi važan element *ledger*-a, *blockchain*, je zapisnik transakcija koje su dovele do trenutnog izgleda stanja sredstava.

Kao što smo već ranije nagovijestili, u Fabricu se za izgled *blockchain*-a ne moraju brinuti svi čvorovi zajedno, nego taj zadatak prepuštaju posebnim servisima koji odlučuju o kreiranju i dodavanju novih blokova na *blockchain*. Za njih se koristi termin *ordering service*, a u svakom kanalu mora biti prisutan bar jedan takav čvor. *Ordering service* čvorovi se ističu po tome što ne održavaju bazu stanja sredstava, nego čuvaju samo *blockchain*.

Uz *ordering service*, životni vijek jedne transakcije izgleda ovako: jedan čvor, ili klijentska aplikacija spojena na njega, predloži transakciju. Taj ju čvor prosljeđuje onim čvorovima za koje je politikom propisano da ju moraju odobriti kako bi bila prihvaćena u sustavu. Ako su odobrenja prikupljena, predlagatelj transakcije ju, zajedno s odobrenjima, prosljeđuje *ordering service*-u, koji može kreirati blok kad prikupi dovoljno odobrenih transakcija. Konačno, gotovi se blok pakira i distribuira svim čvorovima, koji onda pregledavaju sve transakcije u bloku kako bi potvrdili da su sve transakcije valjane i kako bi dodali blok u svoju lokalnu kopiju *blockchain*-a.

Za diseminaciju informacija o izgledu *ledger*-a, kao i za otkrivanje čvorova i kanala u mreži, koristi se *gossip protocol* (protokol ogovaranja).[2]

Svaki čvor u mreži čuva svoju lokalnu kopiju *ledger*-a, ali zbog strogo propisanog načina dodavanja blokova pomoću *ordering service*-a, sve kopije *ledger*-a trebale bi izgledati jednako.

Korištenje Fabrica

Kako bi se pokrenuo novi Fabric projekt, potrebno je prvo pokrenuti *blockchain* mrežu. Konfiguracija mreže se definira kroz nekoliko osnovnih `.yaml` datoteka. Za rad s Fabric mrežom dostupni su gotovi alati `cryptogen` i `configtxgen` koji će iz konfiguracijskih datoteka stvoriti kriptografske ključeve za digitalni potpis, blok postanka za *blockchain* te organizacije te pripadajuće čvorove i korisnike mreže. Potom se mreža pokreće koristeći alat Docker Compose.[3]

Kad je mreža podignuta, mogu se definirati kanali između stvorenih čvorova. Zatim se na svakom kanalu definiraju sredstva koja će se razmjenjivati, instaliraju pametni ugovori te započinju *ledger*-i. Pametni ugovori i sredstva se definiraju koristeći programski kod, a podržani programski jezici su Go, JavaScript, TypeScript i Java. Za pakiranje koda, instaliranje na svaki čvor i usvajanje definiranog pametnog ugovora na cijelom kanalu, dostupan je dodatni Fabricov gotovi alat `peer`. Isti alat može se koristiti i za obavljanje transakcija kroz naredbeni redak.

Konačno, ako je mreža pokrenuta i kanal s pametnim ugovorom uspješno definiran, na mrežu se mogu spajati klijentske aplikacije. Za programiranje aplikacija podržani su isti programski jezici kao i za *chaincode*. U aplikaciji se definira koji čvor mreže se koristi kao poslužitelj (odnosno, kojem čvoru je ta klijentska aplikacija klijent).

Najvažniji Fabricovi alati su prikazani tablicom 3.1.

Što se tiče implementacije pametnih ugovora i klijentskih aplikacija, za isto su dostupna aplikacijska programska sučelja (engl. *application programming interface*, API) u svim ranije navedenim programskim jezicima. O API-ju za jezik Java može se pročitati na adresi `https://hyperledger.github.io/fabric-chaincode-java/`. API za klijentske aplikacije zove se Fabric Gateway, a informacije o Java API-ju mogu se naći na adresi `https://hyperledger.github.io/fabric-gateway/`.

[2]Protokol ogovaranja je protokol kojim se osigurava replikacija podataka u distribuiranom sustavu na način da, u pravilnim vremenskim razmacima, svaki čvor nasumično bira po jednog susjeda kojemu šalje najnovije informacije koje ima. Frekvencija slanja informacija je unaprijed fiksirana.

[3]Docker Compose je alat koji olakšava definiranje i dijeljenje aplikacija koje djeluju u više različitih okruženja odjednom. Praktičan je pri razvoju distribuiranih aplikacija jer, koristeći međusobno izolirane *container*-e, može simulirati rad više čvorova distribuirane mreže na istom računalu.

`peer`	za administraciju čvorova u mreži, vezano za *chaincode*, kanale, pokretanje novih čvorova i sl.;
`osnadmin`	za administraciju čvora na kojem djeluje *ordering service*;
`configtxgen`	omogućuje kreiranje i pregled artefakata vezanih za konfiguraciju kanala;
`configtxlator`	prevodi Fabricove strukture podataka iz Protobuf[4] formata u JSON i obratno;
`cryptogen`	se koristi samo u testnim okruženjima za kreiranje ključeva za digitalni potpis;
`ledgerutil`	služi za otkrivanje greške ako dođe do grananja *blockchain*-a;
`discover`	pomaže u otkrivanju konfiguracije mreže (izlistava čvorove mreže, svojstva kanala i odobravatelje za *chaincode*);
`fabric-ca`	je skup alata za upravljanje Fabricovom uslugom za dodjeljivanje vjerodajnica.

Tablica 3.1: Alati Hyperledger Fabrica

Budući da se klijentske aplikacije i *chaincode* implementiraju koristeći programska sučelja, ne moraju biti pisani u istom programskom jeziku. Primjerice, moguće je napisati *chaincode* u jeziku Go i na mrežu spojiti klijentsku aplikaciju pisanu u jeziku Java.

3.3 Implementacija sustava za izdavanje diploma

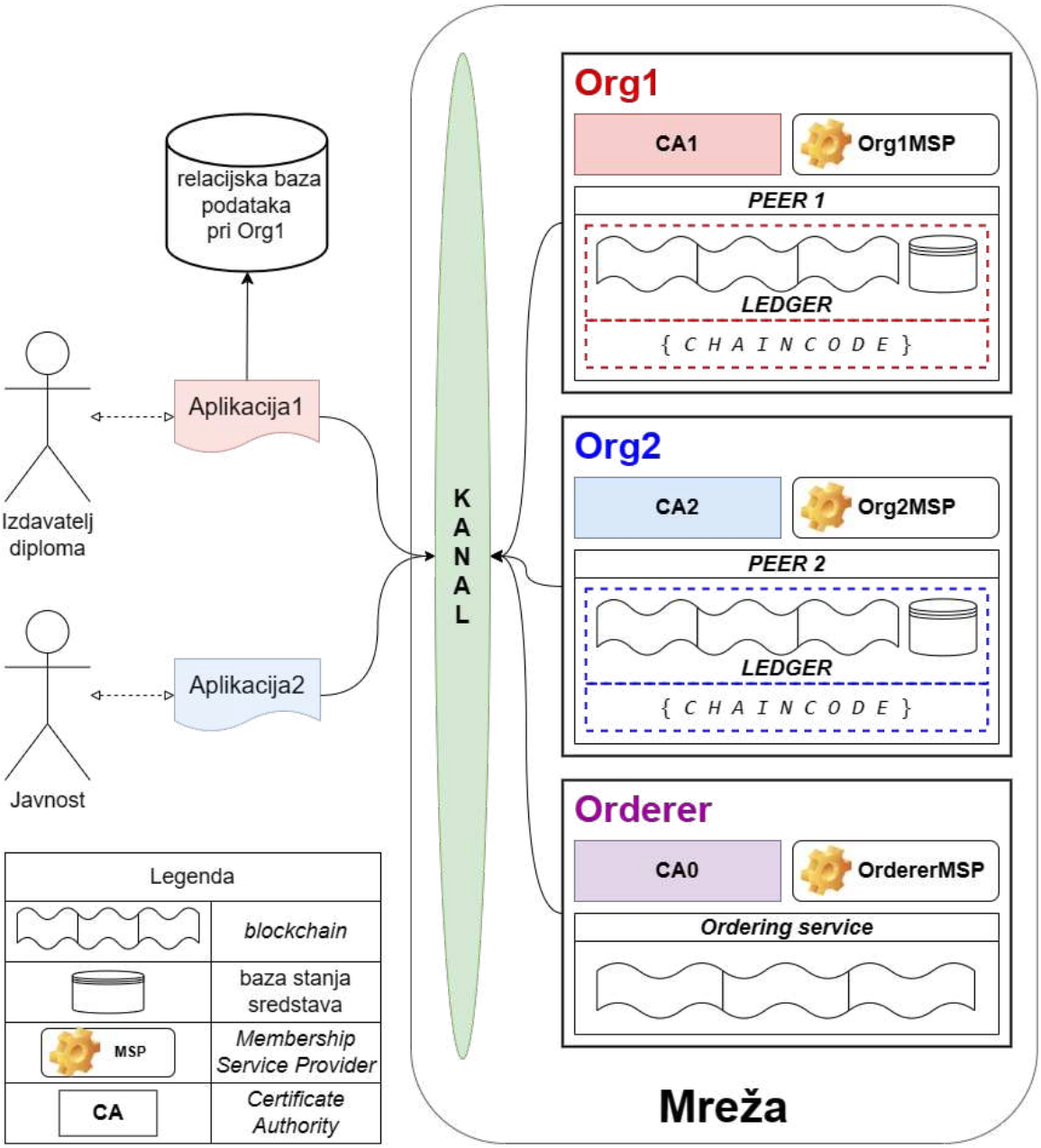
Koristeći Hyperledger Fabric kako je opisano u odjeljku 3.2, implementiran je sustav pomoću kojega se na *blockchain* mogu spremati diplome fakulteta, tako da su javno dostupne za pregled, a kreirati ih mogu samo ovlaštene osobe. Na adresi `https://github.com/mirnana/bureauchain` dostupan je repozitorij u kojem se nalazi projekt.

Slika 3.1 prikazuje shemu cijelog sustava za izdavanje diploma, a detalje o sustavu iznosimo u nastavku odjeljka.

Mreža

Detalji o konfiguraciji Fabric mreže nisu u fokusu ovog rada, stoga je upotrijebljena Fabricova testna mreža. Naime, Fabric uz sve svoje alate i dokumentaciju nudi i veći broj primjera korištenja Fabrica. Između ostalog, dostupan je i niz skripti i konfiguracijskih datoteka koje omogućuju brzo i elegantno stvaranje mreže s *ordering service*-om (na slici

[4]Protobuf (Protocol Buffers) je (binarni) format podataka koji se rabi za serijalizaciju strukturiranih podataka.



Slika 3.1: Shema sustava

3.1: `Orderer`) i dvije organizacije (na slici 3.1: `Org1` i `Org2`), od kojih svaka ima po jedan čvor.

Kao što je na slici prikazano, svaka organizacija ima vlastite ustanove za izdavanje vjerodajnica i za odobravanje pristupa mreži na temelju vjerodajnice (CA i MSP, redom). Isto tako, svaka organizacija u ovom slučaju ima po jedan *peer*, koji održava svoju kopiju *ledger*-a i ima instaliran *chaincode*.

Sadržaj mape koja čini testnu mrežu prije pokretanja, s tim da su izostavljeni dijelovi koji za ovaj sustav nisu potrebni, je sljedeći:

```
test-network
|-- compose
|   |-- compose-ca.yaml
|   |-- compose-couch.yaml
|   |-- compose-test-net.yaml
|   |-- docker
|       |-- docker-compose-ca.yaml
|       |-- docker-compose-couch.yaml
|       |-- docker-compose-test-net.yaml
|       |-- peercfg
|           |-- core.yaml
|-- configtx
|   |-- configtx.yaml
|-- monitordocker.sh
|-- network.sh
|-- organizations
|   |-- ccp-generate.sh
|   |-- ccp-template.json
|   |-- ccp-template.yaml
|   |-- cryptogen
|   |  |-- crypto-config-orderer.yaml
|   |  |-- crypto-config-org1.yaml
|   |  |-- crypto-config-org2.yaml
|   |-- fabric-ca
|       |-- ordererOrg
|       |   |-- fabric-ca-server-config.yaml
|       |-- org1
|       |   |-- fabric-ca-server-config.yaml
|       |-- org2
|       |   |-- fabric-ca-server-config.yaml
|       |-- registerEnroll.sh
|-- README.md
|-- scripts
|   |-- ccutils.sh
|   |-- configUpdate.sh
|   |-- createChannel.sh
|   |-- deployCCAAS.sh
|   |-- deployCC.sh
|   |-- envVar.sh
|   |-- pkgcc.sh
|   |-- setAnchorPeer.sh
|   |-- utils.sh
|-- system-genesis-block
```

Skripta `network.sh` može se upotrijebiti za pokretanje mreže, stvaranje kanala, instalaciju pametnih ugovora i zaustavljanje cijele mreže. Pozivanjem prikladnih naredbi, automatski se generiraju vjerodajnice za korisnike oba čvora mreže, sukladno zadanoj konfiguraciji.

Pametni ugovor

Chaincode je pisan u programskom jeziku Java.

Općenito, svaki *chaincode* je klasa koja implementira sučelje `ContractInterface`. Također je potrebno dodati anotacije: `@Transaction` za metode *chaincode*-a koje implementiraju neku vrstu transakcije, `@DataType` za klase koje predstavljaju sredstva čije se transakcije prate, `@Property` za atribute tih klasa.

Sredstvo čije se transakcije prate u ovoj mreži su objekti naziva `Diploma`. Kako bi zapis transakcija na *ledger* bio uspješan, potrebno je osigurati deterministčnu serijalizaciju objekata u JSON format. Naime, JSON sam po sebi nije deterministčan tip podataka pa je potrebno posebno povesti računa da poredak atributa unutar JSON objekta bude fiksan, za što se može upotrijebiti knjižnica `Genson`. Stoga se parametrima konstruktora za objekt `Diploma` dodaje anotacija `@JsonProperty` i implementiraju se tzv. *getter*-i i *setter*-i[5] za sve atribute.

```
@Property() private final String diplomaID;

@Property() private String nationalID;

@Property() private String firstName;
@Property() private String lastName;
@Property() private String dateOfBirth;
@Property() private String placeOfBirth;
@Property() private String dateOfIssue;

@Property() private String institution;
@Property() private String course;
@Property() private String level;
@Property() private String degree;
```

Isječak koda 3.1: Atributi klase Diploma

Metode koje omogućuju razne vrste transakcija zadane su u klasi `DiplomaContract`. Prvi parametar svake takve metode mora biti objekt klase `Context`, koji prenosi kontekst transakcije pri njenom izvršavanju. Pozivom metode `getStub` tog objekta može se pristupiti *ledger*-u radi čitanja istoga ili dodavanja nove transakcije.

[5] Općenito, *getter* za neki atribut je funkcija koja vraća vrijednost tog atributa, a *setter* atributa je funkcija koja postavlja vrijednost tog atributa.

Navedimo dvije osnovne metode za manipulaciju diplomama — metodu za stvaranje nove diplome i metodu za čitanje diplome sa zadanim identifikatorom.

```
@Transaction(intent = Transaction.TYPE.EVALUATE)
public Diploma readDiploma(final Context ctx, final String diplomaID) {
    if (!diplomaExists(ctx, diplomaID)) {
            throw new ChaincodeException("The diploma " + diplomaID
                + " does not exist");
    }
    String diplomaJSON = ctx.getStub().getStringState(diplomaID);
    Diploma diploma = genson.deserialize(diplomaJSON, Diploma.class);
    return diploma;
}

@Transaction(intent = Transaction.TYPE.SUBMIT)
public void createDiploma(Context ctx
                        , String diplomaID
                        , String nationalID
                        , String firstName
                        , String lastName
                        , String dateOfBirth
                        , String placeOfBirth
                        , String dateOfIssue
                        , String institution
                        , String course
                        , String level
                        , String degree) {
    if (diplomaExists(ctx, diplomaID)) {
        throw new ChaincodeException("The diploma " + diplomaID + "
           already exists");
    }
    Diploma diploma = new Diploma(diplomaID
                                , nationalID
                                , firstName
                                , lastName
                                , dateOfBirth
                                , placeOfBirth
                                , dateOfIssue
                                , institution
                                , course
                                , level
                                , degree);
    String sortedJSON = genson.serialize(diploma);
    ctx.getStub().putStringState(diplomaID, sortedJSON);
}
```

Isječak koda 3.2: Osnovne metode za manipulaciju diplomama

U ovim metodama koristi se pomoćna metoda za otkrivanje postoji li neka diploma na *ledger*-u i pomoćni objekt klase `Genson` za serijalizaciju objekata u JSON format i stvaranje objekata iz stringova u JSON formatu (tzv. deserijalizacija).

```
private final Genson genson = new Genson();

@Transaction(intent = Transaction.TYPE.EVALUATE)
public boolean diplomaExists( final Context ctx
                            , final String diplomaID) {
    String diplomaJSON = ctx.getStub().getStringState(diplomaID);
    return (diplomaJSON != null && !diplomaJSON.isEmpty());
}
```

Isječak koda 3.3: Pomoćni metoda i objekt

Ako se Fabric mreža pokrene s dokumentskom bazom podataka CouchDB kao bazom stanja sredstava, onda je moguće čitati diplome prema ostalim atributima. U tu svrhu se zadaju CouchDB indeksi, kako bi upiti nad bazom bili efikasniji. Pri postavljanju upita koristi se CouchDB selektor, u kojem se navode vrijednosti onih atributa prema kojima pretražujemo i indeks koji se treba koristiti. Navodimo jedan od implementiranih indeksa i pripadnu metodu za čitanje diploma.

```
{
  "index": {
    "fields": [
      "firstName",
      "lastName"
    ]
  },
  "ddoc": "indexNameDoc",
  "name": "indexName",
  "type": "json"
}
```

Isječak koda 3.4: Indeks za pretragu po imenu vlasnika

```
@Transaction(intent =
   Transaction.TYPE.EVALUATE)
public Diploma[] queryDiplomasByName(
          final Context ctx
        , final String firstName
        , final String lastName) throws
           Exception,
           UnsupportedOperationException {

    String selector = String.format(
            "{\"selector\":" +
                "{\"firstName\":\"%s\"," +
                "\"lastName\":\"%s\"}, " +
            "\"use_index\":" +
                "[\"/indexNameDoc\", " +
                "\"indexName\"]}"

            , firstName
            , lastName);
    return getQueryResult(ctx, selector);
}
```

Isječak koda 3.5: Metoda za pretragu po imenu vlasnika

Za sve metode koje omogućuju pretragu po atributima osim identifikatora, implementirana je jedna metoda koja se unutar njih poziva.

```
private Diploma[] getQueryResult( final Context ctx
                                , final String selector) throws
                                    Exception,
                                    UnsupportedOperationException {

    List<Diploma> queryResults = new ArrayList<Diploma>();

    try (QueryResultsIterator<KeyValue> results =
        ctx.getStub().getQueryResult(selector)) {

        for (KeyValue result : results) {
            if (result.getStringValue() == null ||
                result.getStringValue().length() == 0) {
                continue;
            }
            Diploma diploma =
                genson.deserialize(result.getStringValue(),
                Diploma.class);
            queryResults.add(diploma);
        }
    }

    return queryResults.toArray(new Diploma[0]);
}
```

Isječak koda 3.6: Pomoćna metoda za čitanje prema atributu

Metoda kojom se mijenjaju atributi postojeće diplome, `updateDiploma`, vrlo je slična već navedenoj `createDiploma`; razlikuju se samo u tome što se za `updateDiploma` diže iznimka ako diploma s danim identifikatorom *ne postoji*. Preostaje prikazati metode za dohvat svih postojećih diploma i za brisanje diplome dohvaćene po identifikatoru. Napomenimo da brisanje diplome ne znači da je svaki trag o njoj izbrisan s *blockchain*-a. Naprotiv, na *blockchain*-u jednom zapisane informacije zauvijek ostaju sačuvane, a zapis se briše jedino iz baze stanja sredstava.

```
@Transaction(intent = Transaction.TYPE.EVALUATE)
public String getAllDiplomas(final Context ctx) {

    List<Diploma> queryResults = new ArrayList<Diploma>();
    QueryResultsIterator<KeyValue> results =
        ctx.getStub().getStateByRange("", "");

    for (KeyValue result : results) {
        Diploma diploma = genson.deserialize(result.getStringValue(),
            Diploma.class);
```

```
            System.out.println(diploma);
            queryResults.add(diploma);
        }

        final String response = genson.serialize(queryResults);
        return response;
    }

    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public void deleteDiploma(Context ctx, String diplomaID) {

        if (!diplomaExists(ctx, diplomaID)) {
            throw new ChaincodeException("The diploma " + diplomaID + "
                does not exist");
        }

        ctx.getStub().delState(diplomaID);
    }
```

Isječak koda 3.7: Metode za dohvat svih diploma i za brisanje diplome

Klijentske aplikacije

Na Fabricovu mrežu na kojoj je instaliran *chaincode* opisan u prethodnoj sekciji mogu se spajati dvije klijentske aplikacije. Obje su aplikacije napisane u programskom jeziku Java.

Prva, na slici 3.1 prozvana `Aplikacija1`, je dizajnirana da ju koriste osobe ovlaštene da kreiraju, brišu, čitaju i mijenjaju diplome; primjerice, djelatnici ureda za studente na nekom fakultetu. Takvi korisnici na slici 3.1 označeni su `Izdavatelj diploma`.

Druga je aplikacija namijenjena za javnu uporabu i dopušta samo čitanje diploma, koristeći više različitih kriterija za pretragu. Pritom su sve metode za čitanje jednako implementirane za obje aplikacije. Ova aplikacija je na slici 3.1 označena `Aplikacija2`, a njen korisnik je `Javnost`.

`Aplikacija1` i `Aplikacija2` *pripadaju* organizacijama `Org1` i `Org2`, redom, a obje se *spajaju* na kanal putem Gateway servisa.

Autentikacija korisnika za obje aplikacije nije u fokusu ovog rada i nije implementirana, nego su vjerodajnice za spajanje na čvorove Fabric mreže direktno učitane.

Sve metode koje omogućuju obavljanje transakcija na *ledger*-u pozivaju metode pametnog ugovora iz prethodne sekcije preko Fabric Gateway API-ja. Kao primjer, navodimo jednu od metoda za dohvat diploma, i to pretragom po imenu vlasnika diplome. Metoda iz pametnog ugovora koja se pritom poziva je prikazana u isječku koda 3.5.

```
private String readDiplomaByName( String firstName, String lastName)
        throws GatewayException {
```

```
    var result = contract.evaluateTransaction("queryDiplomasByName"
                                              , firstName, lastName);
    return prettyJson(result);
}
```

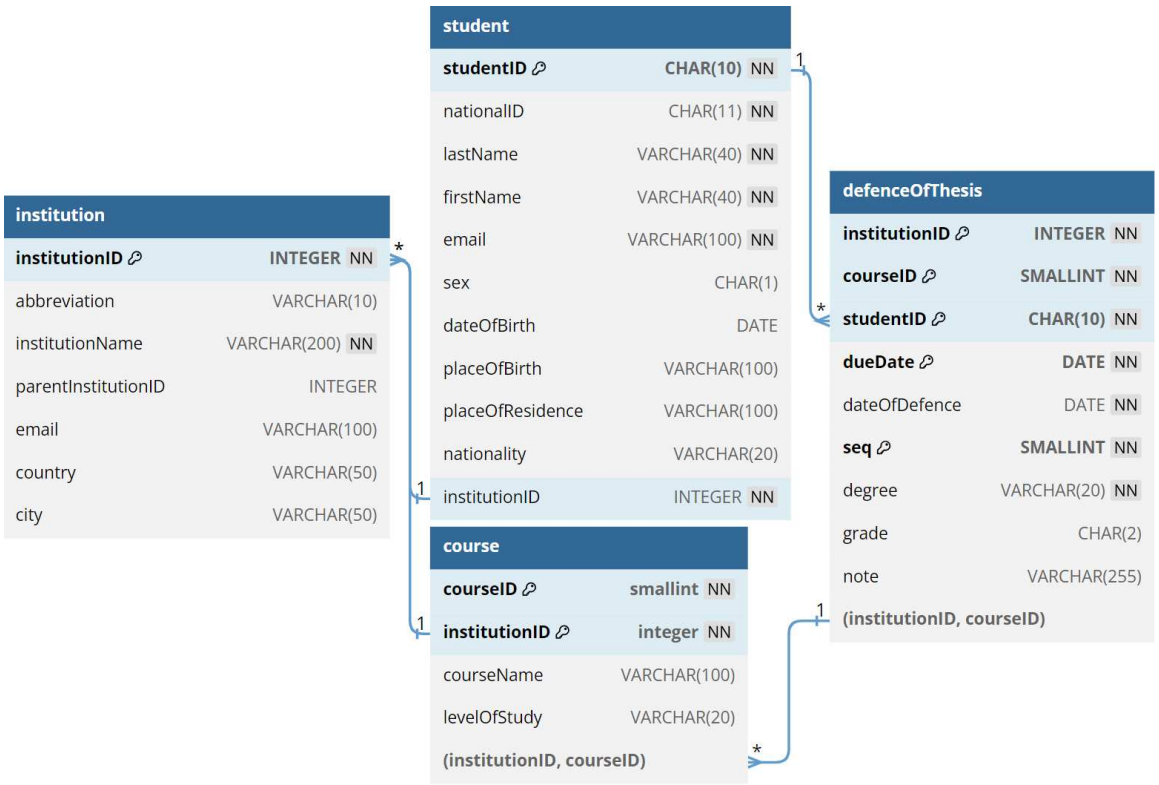
Isječak koda 3.8: Metoda klijentske aplikacije za čitanje diploma po imenu vlasnika

Aplikacije se koriste putem komandno-linijskog sučelja, a diplome se ispisuju u JSON formatu. Kako bi takav ispis bio čitljiviji, koriste se pomoćne metode za formatiranje.

```
private String prettyJson(final byte[] json) {
    return prettyJson(new String(json, StandardCharsets.UTF_8));
}

private String prettyJson(final String json) {
    var parsedJson = JsonParser.parseString(json);
    return gson.toJson(parsedJson);
}
```

Isječak koda 3.9: Pomoćne metode za uljepšavanje ispisa u JSON formatu



Slika 3.2: Shema baze podataka

Ostale metode za čitanje diploma, kao i metode za ažuriranje ili brisanje, implementiraju se vrlo slično kao metoda 3.8 pa ih ovdje ne navodimo. Metode za kreiranje diploma prvo rade upite nad relacijskom bazom podataka i onda pozivaju metodu `createDiploma` *chaincode*-a. Shema baze podataka dizajnirane u sklopu projekta dana je slikom 3.2.

Pritom tablica `institution` opisuje organizacijsku jedinicu u sustavu visokih učilišta, `course` opisuje studij na visokom učilištu, `student` opisuje nekog studenta koji je upisan na neko visoko učilište, a `defenceOfThesis` prikazuje jednu obranu rada kojom student zaključuje neku razinu školovanja. U zadnjoj tablici atribut `dueDate` označava datum roka predaje rada, a `seq` i `dateOfDefence` označavaju redni broj i datum pojedinog pokušaja obrane tog rada; smatra se da je obrana uspješna ako je upisana ocjena, `grade`.

U metodama za kreiranje diploma koriste se pomoćne metode za izvršavanje upita nad bazom prikazane u isječku 3.10. Prikazane metode služe za vađenje podataka potrebnih za izradu diplome, i to iz tablica `student`, `course` i `institution`, redom.

```
private Map<String, String> fromStudentPrepStmtResults(String
    studentID) throws SQLException, Exception {
    Connection c = DriverManager.getConnection(DB_URL, DB_USERNAME,
        DB_PASSWORD);
    PreparedStatement stmt = c.prepareStatement(
        "SELECT nationalID, firstName, lastName, dateOfBirth, " +
        "       placeOfBirth, institutionID " +
        "  FROM student " +
        " WHERE studentID = ?;");
    stmt.setString(1, studentID);
    ResultSet rs = stmt.executeQuery();

    int rowCount = 0;
    if (rs.last()) {
        rowCount = rs.getRow();
    }
    if (rowCount == 0) {
        throw new Exception("Student with studentID " + studentID + "
            does not exist");
    }

    rs.first(); // one row expected as studentID is the primary key
    Map<String, String> attributes = new HashMap<>();
    attributes.put("nationalID", rs.getString("nationalID"));
    attributes.put("firstName", rs.getString("firstName"));
    attributes.put("lastName", rs.getString("lastName"));
    attributes.put("dateOfBirth", rs.getString("dateOfBirth"));
    attributes.put("placeOfBirth", rs.getString("placeOfBirth"));
    attributes.put("institutionID", rs.getString("institutionID"));

    c.close();
```

```
    return attributes;
}

private Map<String, String> fromCoursePrepStmtResults(Integer courseID,
    Integer institutuionID) throws SQLException {
    Connection c = DriverManager.getConnection(DB_URL, DB_USERNAME,
        DB_PASSWORD);
    PreparedStatement stmt = c.prepareStatement(
        "SELECT courseName, levelOfStudy " +
        "  FROM course " +
        " WHERE courseID = ?" +
        "   AND institutionID = ?;");
    stmt.setInt(1, courseID);
    stmt.setInt(2, institutuionID);
    ResultSet rs = stmt.executeQuery();

    rs.first();
    Map<String, String> attributes = new HashMap<>();
    attributes.put("courseName", rs.getString("courseName"));
    attributes.put("levelOfStudy", rs.getString("levelOfStudy"));

    c.close();
    return attributes;
}

private Map<String, String> fromInstitutionPrepStmtResults(Integer
    institutionID) throws SQLException {
    Connection c = DriverManager.getConnection(DB_URL, DB_USERNAME,
        DB_PASSWORD);
    PreparedStatement stmt = c.prepareStatement(
        "SELECT institutionName, parentInstitutionID " +
        "  FROM institution " +
        " WHERE institutionID = ?;");
    stmt.setInt(1, institutionID);
    ResultSet rs = stmt.executeQuery();

    rs.first();
    Map<String, String> attributes = new HashMap<>();
    attributes.put("institutionName", rs.getString("institutionName"));
    attributes.put("parentInstitutionID",
        rs.getString("parentInstitutionID"));

    c.close();
    return attributes;
}
```

Isječak koda 3.10: Izvršavanje upita nad bazom

Prva metoda za kreiranje diploma prima identifikator studenta (primjerice Jedinstveni matični broj akademskog građanina — JMBAG u Hrvatskoj) i prema njemu iz baze čita osobne podatke studenta i pregledava postoji li zapis o njegovim obranama završnog rada (za bilo koju razinu studija). Ako postoji obrana, vade se podaci o visokom učilištu i studiju, a na posljetku se na *ledger*-u provjerava je li pripadna diploma već izdana. Pod uvjetom da diploma s prikupljenim odrednicama već ne postoji, kreira se nova diploma i dodaje na *ledger*.

```
private void createDiplomaByStudentID(String studentID) throws
    SQLException, Exception {
     try {

         System.out.println("... querying the local relational database
             ...");

         Map<String, String> studentAttributes = new
             HashMap(fromStudentPrepStmtResults(studentID));
         String nationalID = studentAttributes.get("nationalID");
         String firstName = studentAttributes.get("firstName");
         String lastName = studentAttributes.get("lastName");
         String dateOfBirth = studentAttributes.get("dateOfBirth");
         String placeOfBirth = studentAttributes.get("placeOfBirth");
         String institutionID = studentAttributes.get("institutionID");
         String institutionID2 = institutionID;

         String institution = "";
         String parentInstitutionID = "";
         boolean firstIteration = true;
         do {
             Map<String, String> institutionAttributes = new HashMap<>(
                 fromInstitutionPrepStmtResults(
                     Integer.parseInt(institutionID)));

             if (firstIteration) {
                 firstIteration = false;
             } else {
                 institution += ", ";
             }

             institution += institutionAttributes.get("institutionName");
             institutionID = parentInstitutionID
                           = institutionAttributes
                                 .get("parentInstitutionID");
         } while (parentInstitutionID != null);

         Connection c = DriverManager.getConnection(DB_URL, DB_USERNAME,
             DB_PASSWORD);
```

```
        PreparedStatement stmt = c.prepareStatement(
            "SELECT courseID, degree " +
            "  FROM defenceOfThesis " +
            " WHERE institutionID = ? " +
            "   AND studentID = ? " +
            "   AND grade IS NOT NULL " +
            "ORDER BY dueDate DESC, dateOfDefence DESC, seq DESC;"
        );
        stmt.setInt(1, Integer.parseInt(institutionID2));
        stmt.setString(2, studentID);
        ResultSet rs = stmt.executeQuery();

        int rowCount = 0;
        if (rs.last()) {
            rowCount = rs.getRow();
            rs.beforeFirst();
        }
        if (rowCount == 0) {
            throw new Exception("Student with studentID " + studentID +
               " has not defended any theses");
        }
        while(rs.next()){
            String courseID = rs.getString("courseID");
            String degree = rs.getString("degree");

            Map<String, String> courseAttributes = new HashMap<>(
                fromCoursePrepStmtResults(
                    Integer.parseInt(courseID),
                       Integer.parseInt(institutionID2)));
            String courseName   = courseAttributes.get("courseName");
            String levelOfStudy = courseAttributes.get("levelOfStudy");

            System.out.println("... data from relational database
               retreived ...");

            System.out.println("... checking if diploma already exists
               ...");
            String fromLedger = readDiplomaByPrimKey( nationalID
                                                    , institution
                                                    , courseName
                                                    , levelOfStudy);
            if (!fromLedger.equals("[]")) {
                System.out.println("Diploma with the given parameters
                   already exists: " + fromLedger);
                continue;
            }
```

```
            String diplomaID = "diploma" + Instant.now().toEpochMilli();
            contract.submitTransaction("createDiploma"
                                       , diplomaID
                                       , nationalID
                                       , firstName
                                       , lastName
                                       , dateOfBirth.toString()
                                       , placeOfBirth
                                       , LocalDate.now().toString()
                                       , institution
                                       , courseName
                                       , levelOfStudy
                                       , degree);
            System.out.println("Successfully created new diploma " +
                diplomaID);
        }

        c.close();

    } catch (Exception e) {
        System.out.println("ERROR while creating diploma by ID: " +
            e.getMessage());
    }
}
```

Isječak koda 3.11: Metoda za kreiranje diplome temeljem identifikatora studenta

Druga metoda za kreiranje diploma koristi nešto drukčiji pristup. Metoda prima datum obrane (`dateOfDefence`), čita zapise o uspješnim obranama radova koje su se dogodile na taj datum i za svaki od njih vadi podatke o studentu, studiju i ustanovi. Ako diploma s istim odrednicama već ne postoji, kreira novu diplomu. Budući da se implementira analogno kao prva metoda, ne navodimo čitavo tijelo metode, nego u isječku koda 3.12 prikazujemo samo upit koji se izvršava nad bazom podataka.

```
Connection c = DriverManager.getConnection(DB_URL, DB_USERNAME,
    DB_PASSWORD);
PreparedStatement stmt = c.prepareStatement(
    "SELECT institutionID, courseID, studentID, degree " +
    "  FROM defenceOfThesis " +
    " WHERE dateOfDefence = ? " +
    "   AND grade IS NOT NULL;"
);
stmt.setString(1, dateOfDefence);
ResultSet rs = stmt.executeQuery();
```

Isječak koda 3.12: Upit za kreiranje diploma na temelju datuma obrane

Korištenje razvijene klijentske aplikacije

Po upoznavanju s detaljima implementacije cijelog projekta, pogledajmo i jedan mogući tijek korištenja aplikacije. Promatramo aplikaciju koja nudi kreiranje, brisanje, mijenjanje i čitanje diploma (`Aplikacija1` sa slike 3.1).

Korisnički unos označavat ćemo u svim primjerima simbolom >.

Kad se aplikacija pokrene, ispisuje se ponuda funkcionalnosti:

```
Enter: r to read diploma by ID
       a to read all diplomas
       p to read diploma by the 'primary key'
       n to read diploma by the owner's name
       i to read diploma by the owner's national ID
       s to create diplomas by student ID
       t to create diplomas by date of defence of thesis
       u to update a diploma
       d to delete a diploma
       x to exit
```

Potom možemo pokušati kreirati diplomu temeljem datuma obrane. Odaberemo li datum na koji nije bilo obrana, dobijemo sljedeće:

```
>t
 Insert date in format YYYY-MM-dd:
>2020-01-01
 ... querying the local relational database ...
 ERROR while creating diploma by date of defence:
 No defences found on date 2020-01-01
```

Ako pak odaberemo korektan datum, obaviješteni smo o uspješno kreiranim diplomama:

```
>t
 Insert date in format YYYY-MM-dd:
>2023-06-15
 ... querying the local relational database ...
 ... creating diploma for 22222222322 ...
 Successfully created new diploma diploma1693419617643
 ... creating diploma for 22222222323 ...
 Successfully created new diploma diploma1693419622500
```

Odaberemo li opciju kreiranja diploma po studentskom identifikatoru, ali unesemo nepostojeći identifikator, javlja se slična greška:

```
>s
 Insert student ID:
>1111111111
 ... querying the local relational database ...
 ERROR while creating diploma by ID:
 Student with studentID 1111111111 does not exist
```

Kreiranje po studentskom identifikatoru također ne prolazi ako student nije obranio nijedan rad:

```
>s
 Insert student ID:
>0000000001
 ... querying the local relational database ...
 ERROR while creating diploma by ID:
 Student with studentID 0000000001 has not defended any theses
```

Pogledajmo i primjer uspješne izrade diplome (ili, po potrebi, više njih) prema identifikatoru.

```
>s
 Insert student ID:
>0000000010
 ... querying the local relational database ...
 ... data from relational database retreived ...
 ... checking if diploma already exists ...
 Successfully created new diploma diploma1693420128018

>s
 Insert student ID:
>0000000000
 ... querying the local relational database ...
 ... data from relational database retreived ...
 ... checking if diploma already exists ...
 Successfully created new diploma diploma1693419706988
 ... data from relational database retreived ...
 ... checking if diploma already exists ...
 Successfully created new diploma diploma1693419709913
```

Brisanje diplome koja postoji i diplome koja ne postoji rezultira sljedećim ponašanjima:

```
>d
 Insert diploma ID:
```

```
>diploma1693419709913
 Delete successful

>d
 Insert diploma ID:
>diploma1693419709913
 ERROR while deleting diploma:
 chaincode response 500, The diploma diploma1693419709913 does not exist
```

Ažuriranje diplome može proteći na sljedeći način:

```
>u
 Insert diplomaID:
>diploma1693419622500
 Insert nationalID:
>22222222323
 Insert firstName:
>Dora
 Insert lastName:
>T
 Insert dateOfBirth:
>1999-01-26
 Insert placeOfBirth:
>Metković
 Insert dateOfIssue:
>2023-08-28
 Insert institution:
>Zagrebačka škola ekonomije i managementa
 Insert course:
>Ekonomija i management
 Insert level:
>preddiplomski
 Insert degree:
>MBA
 Update successful
```

Za kraj, pogledajmo čitanje diploma po raznim kriterijima pretrage. Po imenu:

```
>n
 Insert first name:
>Ana
 Insert last name:
```

```
>Š
 [
   {
     "firstName": "Ana",
     "lastName": "Š",
     "institution": "Farmaceutsko-biokemijski fakultet,
                     Sveučilište u Zagrebu",
     "placeOfBirth": "Zagreb",
     "nationalID": "22222222322",
     "level": "doktorski",
     "diplomaID": "diploma1693419617643",
     "degree": "dr. sc.",
     "course": "Farmaceutsko-biokemijske znanosti",
     "dateOfBirth": "1998-11-25",
     "dateOfIssue": "2023-08-30"
   }
 ]
```

Pregled po nacionalnom identifikatoru (OIB u Hrvatskoj):

```
>i
 Insert national ID:
>22222222232
 [
   {
     "firstName": "Nicole",
     "lastName": "D",
     "institution": "Matematički odsjek,
                     Prirodoslovno-matematički fakultet,
                     Sveučilište u Zagrebu",
     "placeOfBirth": "Zadar",
     "nationalID": "22222222232",
     "level": "diplomski",
     "diplomaID": "diploma1693420128018",
     "degree": "mag. math.",
     "course": "Financijska i poslovna matematika",
     "dateOfBirth": "1998-12-08",
     "dateOfIssue": "2023-08-30"
   }
 ]
```

Koristeći identifikatore diploma, pretraga ne vraća popis, nego samo jednu diplomu. Stoga, ako se unese identifikator koji ne postoji, javlja se greška.

```
>r
 Insert diploma ID:
>diplomakojanepostoji
 ERROR reading diploma: io.grpc.StatusRuntimeException:
 UNKNOWN: evaluate call to endorser returned error:
 chaincode response 500, The diploma diplomakojanepostoji does not exist

>r
 Insert diploma ID:
>diploma1693419706988
 {
   "firstName": "Ivana",
   "lastName": "R",
   "institution": "Filozofski fakultet Sveučilišta u Zagrebu,
                   Sveučilište u Zagrebu",
   "placeOfBirth": "Slavonski Brod",
   "nationalID": "22222222222",
   "level": "diplomski",
   "diplomaID": "diploma1693419706988",
   "degree": "mag. tal.",
   "course": "Talijanistika",
   "dateOfBirth": "1999-02-06",
   "dateOfIssue": "2023-08-30"
 }
```

Konačno, prikažimo i pretragu prema primarnom ključu diplome. Primarnim ključem smatramo uređenu četvorku nacionalnog identifikatora (OIB-a), visokog učilišta, studija (odnosno smjera na tom visokom učilištu) i akademske razine. Ovakva pretraga koristi se i u metodama za kreiranje novih diploma, pri provjeri postoji li već određena diploma. U aplikaciji, hod korištenja ove funkcionalnosti u slučaju da se unesu podaci diplome koja ne postoji na *ledger*-u je sljedeći (da diploma postoji, lista bi imala točno jedan element, a prikaz bi bio sličan kao u prethodnim primjerima):

```
>p
 Insert national ID:
>01234567890
 Insert institution:
>Fakultet elektrotehnike i računarstva
 Insert course:
>Informacijska i komunikacijska tehnologija
 Insert level:
```

```
>diplomski
  []
```

Bibliografija

[1] I. Blake, G. Seroussi i N. Smart, *Elliptic Curves in Cryptography*, London Mathematical Society Lecture Note Series, Cambridge University Press, 1999.

[2] Quynh Dang, *Secure Hash Standard*, 2015.

[3] Andrej Dujella, *Eliptičke krivulje u kriptografiji*, PMF-MO, Sveučilište u Zagrebu (2013.).

[4] Elad Elrom, *The Blockchain Developer: A Practical Guide for Designing, Implementing, Publishing, Testing, and Securing Distributed Blockchain-based Projects*, 2019., ISBN 978-1-4842-4846-1.

[5] Hyperledger Foundation, *Hyperledger Fabric (dokumentacija)*, 2023., `https://hyperledger-fabric.readthedocs.io/en/release-2.5/`, posjećena u srpnju 2023.

[6] J. Hoffstein, J. Pipher i J.H. Silverman, *An Introduction to Mathematical Cryptography*, Undergraduate Texts in Mathematics, Springer New York, 2008., ISBN 9780387779942.

[7] Holistics, *DBDiagram, alat za crtanje sheme baze podataka*, `https://dbdiagram.io/home`, posjećena u kolovozu 2023.

[8] Mirna Imrović, *Projekt razvijen u sklopu diplomskog rada*, `https://github.com/mirnana/bureauchain`, posjećena u kolovozu 2023.

[9] JGraph Ltd, *draw.io, alat za crtanje raznih vrsta dijagrama*, `https://www.drawio.com/`, posjećena u rujnu 2023.

[10] Robert Manger, *Softversko inženjerstvo*, Element, Zagreb, 2016.

[11] ———, *Distribuirani procesi (Nastavni materijal)*, Sveučilište u Zagrebu, 2023.

[12] Robert Manger i Miljenko Marušić, *Strukture podataka i algoritmi*, Element, Zagreb, 2014.

[13] A.J. Menezes, J. Katz, P.C. van Oorschot i S.A. Vanstone, *Handbook of Applied Cryptography*, Discrete Mathematics and Its Applications, CRC Press, 1996., ISBN 9781439821916.

[14] A. Narayanan, J. Bonneau, E. Felten, A. Miller i S. Goldfeder, *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*, Princeton University Press, 2016., ISBN 9780691171692.

[15] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini i Yarik Markov, *The first collision for full SHA-1*, (2017.), `https://shattered.io/static/shattered.pdf`.

[16] Gengrui Zhang, Fei Pan, Michael Dang'ana, Yunhao Mao, Shashank Motepalli, Shiquan Zhang i Hans Arno Jacobsen, *Reaching Consensus in the Byzantine Empire: A Comprehensive Review of BFT Consensus Algorithms*, 2022.

Sažetak

U ovom radu bavimo se temom korištenja *blockchain*-a u sustavima javne uprave. Razmatramo ideju da bi *blockchain* tehnologija mogla omogućiti transparentnost izdavanja i manipulacije javnih dokumenata te provjerljivost osobnih dokumenata.

Kako bismo, prije svega, mogli razumjeti što je *blockchain* tehnologija, u prvom poglavlju dajemo pregled osnovnih koncepata matematike i računarstva koji su neophodni za razvoj sustava temeljenog na *blockchain*-u. Objašnjavamo što su kriptografske *hash* funkcije i prezentiramo algoritam za računanje jedne od najraširenijih kriptografskih *hash* funkcija, SHA-256. Potom proučavamo matematički koncept digitalnog potpisa, odnosno kriptografije javnog ključa; konkretno, predstavljamo algoritam za digitalni potpis koji koristi eliptičke krivulje, ECDSA, koji je ujedno i najzastupljeniji u sustavima temeljenim na *blockchain*-u. Proučavamo i računarski koncept distribuiranog konsenzusa, kao i otpornosti računalnog sustava na razne vrste grešaka, budući da ideje iza sustava temeljenih na *blockchain*-u sežu i dublje od pukog korištenja nove strukture podataka.

U drugom poglavlju definiramo *blockchain* kao strukturu podataka, kao i neke druge strukture koje su pritom potrebne. Objašnjavamo zašto se sustavi temeljeni na *blockchain*-u dizajniraju u modelu ravnopravnih partnera (odnosno, kao *peer-to-peer* sustavi). Potom promatramo konvencije za daljnje modeliranje sustava temeljenih na *blockchain*-u, objašnjavajući pritom najpoznatije protokole za postizanje distribuiranog konsenzusa i biranje vođe. Uvodimo koncept pametnih ugovora, koji su često predstavljeni kao inovacija donešena s *blockchain*-om. Diskutiramo zašto se *blockchain*-u nerijetko dodaje apozicija *tehnologija*. Navodimo mogućnosti primjene *blockchain*-a i sustava koji se vode ranije navedenim konvencijama pri korištenju te strukture podataka.

Temu mogućnosti primjene *blockchain*-a, i to u svrhu unapređenja digitalnog poslovanja u javnoj upravi, produbljujemo u posljednjem poglavlju. Posebno komentiramo paralelno korištenje *blockchain*-a i uvriježenih relacijskih baza podataka, te objašnjavamo zašto bi se moglo smatrati da je takva kombinacija tehnologija optimalna, kako za javne organizacije tako i za građane. Dajemo pregled jednostavnog sustava za izradu i manipulaciju fakultetskih diploma, koji je implementiran u sklopu ovog rada. Navedeni sustav ilustrira kako bi se moglo upravljati jednom vrstom dokumenata koji su privatni, a trebaju biti dostupni za javni pregled, koristeći *blockchain*. Za implementaciju sustava korišten je pro-

gramski okvir Hyperledger Fabric pa ukratko predstavljamo i njega i njegove mogućnosti. Potanko navodimo detalje o implementaciji i korištenju klijentskih aplikacija koje se vežu na implementirani sustav temeljen na *blockchain*-u.

Summary

This paper deals with the use of blockchain in public administration systems. We are considering the idea that blockchain technology could enable transparency of issuing and manipulating public documents and the verifiability of personal documents.

First of all, in order to understand what blockchain technology is, in the first chapter we give an overview of the basic concepts of mathematics and computer science that are necessary for the development of a blockchain-based system. We explain what cryptographic hash functions are and present an algorithm for calculating one of the most widespread cryptographic hash functions, SHA-256. Then we study the mathematical concept of digital signature, i.e. public key cryptography; in particular, we present an algorithm for digital signature that uses elliptic curves, ECDSA, which is utilized in most blockchain-based systems. We also study the concept of distributed consensus, as well as types of fault tolerance in computer systems, since the ideas behind blockchain-based systems go deeper than simply using a new data structure.

In the second chapter, we define the blockchain as a data structure, as well as some other structures that are needed therein. We explain why blockchain-based systems are designed as peer-to-peer systems. Then we look at conventions to further model blockchain-based systems, explaining the most famous protocols for reaching distributed consensus and electing the leader of a peer-to-peer system. We introduce the concept of smart contracts, which are often presented as an innovation brought with blockchain. Since blockchain is often described as a technology, we look into the reasons why that is. Some possible use-cases of blockchain and blockchain-based systems following the abovementioned conventions are listed at the end of the chapter.

The topic of the possibility of using blockchain, especially in order to improve digital business in public administration, is deepened in the last chapter. We particularly comment on the parallel use of blockchain and the already established relational databases, and explain why such a combination of technologies could be considered optimal for both public organisations and citizens. We give an overview of a simple system for the creation and manipulation of university degrees, which has been implemented as part of this paper. This system illustrates how one type of documents that is private and should be publicly verifiable can be managed using blockchain. The Hyperledger Fabric framework was used

to implement the system, so we briefly present it and its capabilities. We detail the implementation and use of client applications that connect to the implemented blockchain based system.

Životopis

Rođena sam 1. veljače 1999. u Zagrebu. Godine 2013. završila sam Osnovnu školu „Antun Mihanović" u Slavonskom Brodu, a 2017. prirodoslovno-matematički smjer u Gimnaziji „Matija Mesić" u Slavonskom Brodu.

Godine 2017. sam upisala, a 2021. završila preddiplomski sveučilišni studij Matematika na Prirodoslovno-matematičkom fakultetu u Zagrebu. Potom sam 2021. godine upisala diplomski sveučilišni studij Računarstvo i matematika na istom fakultetu.

Od 2020. godine uz studiranje radim i razne studentske i honorarne poslove, a od 2022. sam zaposlena kao student u Sveučilišnom računskom centru — Srce.